



Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Progetto Finale in **Big Data Engineering**

Fact-Checking through Q&A using Retrieval Augmented Generation

Anno Accademico 23/24

Candidati:

Francesco Pio Manna

matr. M63001485

Matteo Rossi

matr. P37000144

Davide Landolfi

matr. M63001524

Indice

Introduzione	1
1 Raccolta e pre-processing dei dati	2
1.1 Fonti dei dati	2
1.2 Raccolta dei dati	2
1.3 Pre-processing del dataset	3
1.3.1 Estrazione autori	4
1.3.2 Estrazione categoria	5
1.3.3 Estrazione publisher	5
1.3.4 Estrazione keywords	6
1.3.5 Estrazione topic	8
1.4 Preparazione dei dati per il RAG	12
1.4.1 MongoDB	12
1.4.2 Pinecone	13
1.4.3 Sincronizzazione tra i due DB	15
2 Chatbot	18
2.1 Pipeline complessiva	18
2.1.1 LLM	18
2.1.2 RAG	19
3 Analytics	23
3.1 Analytic 1: Distribuzione degli articoli nel tempo	24
3.2 Analytic 2: Analisi delle keywords	25
3.3 Analytic 3: Analisi degli autori	27
3.4 Analytic 4: Visualizzazione modello BERTopic	28

3.4.1	Topics per documento	29
3.4.2	Word Scores per topic	30
4	Sviluppi Futuri	31
4.1	Gestione di più utenti	31
4.2	NER e Graph RAG	31
4.3	Biografia autore	32

Introduzione

Nel contesto attuale, caratterizzato da un rapido flusso di informazioni e dalla diffusione di notizie attraverso molteplici piattaforme digitali, la verifica della bontà delle notizie è diventata una componente cruciale per mantenere l'integrità e la veridicità dell'informazione. Il progetto sviluppato affronta questa sfida utilizzando tecnologie avanzate di intelligenza artificiale, combinando la potenza dei modelli di linguaggio di grandi dimensioni (LLM) con un dataset di articoli di giornale per eseguire il fact-checking delle notizie fornite in input dall'utente.

Il sistema, denominato **Retrieval-Augmented Generation (RAG)**, è progettato per analizzare le affermazioni presenti nelle notizie e verificare la loro accuratezza confrontandole con un ampio corpus di articoli di giornale. Utilizzando tecniche di Natural Language Processing (NLP) e modelli di linguaggio avanzati, il RAG è in grado di comprendere il contesto e il contenuto delle affermazioni, integrando informazioni recuperate dal dataset per generare risposte accurate e contestualizzate.

Il progetto inoltre ha lo scopo di fornire all'utente un'interfaccia per esplorare il dataset, filtrandolo e rilevandone informazioni di interesse

Capitolo 1

Raccolta e pre-processing dei dati

L'obiettivo in questa fase è quello di raccogliere articoli di giornale, comprendenti vari metadata, tra cui autore, categoria, testo e data. Tali articoli saranno utilizzati per realizzare un Chatbot mentre i metadati saranno essenziali per realizzare analytics di vario tipo. Sono state utilizzate delle API per il recupero degli articoli di giornale, un database NoSQL e uno vettoriale per la memorizzazione e tecniche di embedding per trasformare il testo.

1.1 Fonti dei dati

I dati sono stati raccolti grazie alle API fornite da WorldNewsApi. Questa piattaforma dà accesso a migliaia di fonti giornalistiche diverse, di oltre 50 lingue e 150 paesi. Nel caso specifico del progetto, sono state raccolte notizie in lingua italiana.

1.2 Raccolta dei dati

Sfruttando le API di WorldNewsApi, sono stati prelevati 120.000 articoli da 4 testate giornalistiche. In particolare, le testate considerate sono:

- ANSA
- il Sole 24 Ore
- Repubblica
- il Mattino

Tuttavia, a causa delle limitazioni imposte dal piano gratuito del db utilizzato, per le fasi successive sono stati utilizzati circa 31.000 articoli. Gli articoli considerati sono equamente distribuiti su tutto l'asse temporale di pubblicazione delle notizie raccolte, che vanno da marzo 2022 ad aprile 2024. Di seguito è riportato un esempio di notizia restituita da WorldNewsApi.

0	1
id	120180912
title	Toldo lancia l'Inter contro il City, "E' imprevedibile" - Sport - ANSA
text	"Un doppio appuntamento che fa parte nella mia storia personale: ho militato 8 anni nella Fioren...
summary	L'ex portiere: "Tutta Italia fa il tifo anche per la Fiorentina"
url	https://www.ansa.it/sito/notizie/sport/2023/06/05/toldo-lancia-linter-contro-il-city-e-imprevedi...
image	https://www.ansa.it/webimages/img_700/2021/4/24/0f236bed27a82c10f1f2cb20acb0a7d.jpg
publish_date	2023-06-05 11:56:28
author	Redazione ANSA
language	it
source_country	it
sentiment	None

Figura 1.1: Esempio notizia WorldNewsAPI

Come si può notare dalla figura 1.1, vengono restituite le seguenti informazioni:

- **id**: id univoco per la notizia;
- **title**: titolo della notizia;
- **text**: testo della notizia;
- **summary**: riassunto del testo della notizia;
- **url**: url del sito della notizia;
- **image**: url dell'immagine associata alla notizia;
- **publish_date**: data di pubblicazione della notizia;
- **author**: autore della notizia;
- **language**: lingua in cui è scritta la notizia;
- **source_country**: paese in cui è stata scritta la notizia;
- **sentiment**: sentiment associato alla notizia.

Tra le varie informazioni restituite, sono state mantenute solo le colonne relative a *author*, *title*, *publish_date*, *text* e *url*.

Si era ipotizzato inizialmente di non eliminare la colonna *summary* ed utilizzarla per fare l'embedding. Tuttavia, si è deciso di proseguire con la strada del chunking del testo della notizia, motivata anche dal fatto che la colonna *summary* presenta numerosi missing values.

1.3 Pre-processing del dataset

Una volta ottenuto il dataset con 31.000 articoli, sono state effettuate varie operazioni di pre-processing, al fine di preparare i dati alle fasi successive.

1.3.1 Estrazione autori

I dati raccolti presentavano alcuni problemi con la colonna *author*, che in alcuni casi comprendeva la stringa "REDAZIONE ANSA" o altre parole diverse dal nome e cognome dell'autore. Pertanto, è stata utilizzata una regex trovare parole che iniziano con lettere maiuscole seguite da lettere minuscole, separate da uno spazio (ad esempio, "Nome Cognome").

```

1 def extract_author_names(author_list):
2     if not isinstance(author_list, list):
3         return []
4
5     all_matches = []
6     for text in author_list:
7         if not isinstance(text, str):
8             continue
9         # Regex to find words starting with uppercase letters
10        pattern = r'\b[A-Z][a-z]*\s[A-Z][a-z]*\b'
11        # Check for special case "REDAZIONE ANSA"
12        if text.upper() == "REDAZIONE ANSA":
13            all_matches.append("Redazione ANSA")
14            continue
15        # Find all matches
16        matches = re.findall(pattern, text)
17        filtered_matches = [name for name in matches if not re.
18                             search(r'redat', name, flags=re.IGNORECASE)]
19        all_matches.extend(filtered_matches)
20    unique_authors = sorted(set(all_matches))
21    return unique_authors[0]

```

	author
0	[Lucia Tironi]
1	[Roberta Villa]
2	[Carlo Clericetti]
3	[Giovanni Marino]
4	[]

Figura 1.2: Output estrazione autori

Dalla figura 1.2, si può notare come siano presenti alcuni missing values.

1.3.2 Estrazione categoria

Sfruttando la struttura degli URL delle notizie di alcune testate, è possibile estrarre da questi ultimi la categoria di appartenenza della notizia. Tale informazioni può risultare molto utile in fase di costruzione delle analytics. Questa informazione è stata aggiunta come ulteriore colonna.

```

1 def extract_categories(url, category_list=None):
2     if category_list is None:
3         category_list = [
4             "politica",
5             "economia",
6             ...
7             ...
8             ...
9         ]
10
11     pattern = r"https?:/(?:www\.)?[\w.-]+(/[\^/]+)+"
12     match = re.search(pattern, url)
13     if match:
14         path_segments = match.group(0).split('/')
15         categories_found = [segment for segment in path_segments if
16                             segment in category_list]
17         unique_categories = ', '.join(sorted(set(categories_found)))
18     return ''

```

	category
1920	['arte', 'cultura']
20004	['sport']
12583	['mondo']
20414	['politica']
21084	['economia']

Figura 1.3: Output estrazione categoria

1.3.3 Estrazione publisher

Sono stati estratti i nomi delle testate dei giornali dall'URL delle notizie e aggiunti come nuova colonna al dataset.

```

1 def extract_website_name(url):
2     # Remove the 'https://www.' part from the URL
3     website_dict = {
4         'ansa': 'ANSA',
5         'ilfattoquotidiano': 'Il Fatto Quotidiano',
6         'ilsole24ore': 'Il Sole 24 Ore',

```



```

7     'repubblica': 'Repubblica',
8     'ilmattino': 'Il Mattino'
9     }
10    temp = url.split("//www.")[-1].split(".")[0]
11    if temp in website_dict.keys():
12        return website_dict[temp]
13    else:
14        return None

```

1.3.4 Estrazione keywords

In questa fase, si è proceduto all'estrazione delle keywords a partire dal testo, utilizzando KeyBERT.

KeyBERT

KeyBERT è una libreria Python progettata per l'estrazione di keywords da testi utilizzando modelli di embedding di parole basati su BERT (Bidirectional Encoder Representations from Transformers). Sfrutta le potenti capacità di BERT per comprendere il contesto e il significato delle parole in un testo, consentendo un'estrazione più accurata e rilevante delle parole chiave rispetto ai metodi tradizionali.

Per far operare KeyBERT nelle migliori condizioni possibili, sono stati effettuati diversi pre-processing sul testo originale della notizia:

1. **Rimozione dei caratteri accentati:** a causa della difficoltà nell'interpretazione dei caratteri accentati, sono stati rimossi e sostituiti con la loro versione non accentata (à → a);

```

1  def remove_accents(text):
2      accented_chars = '
3
4      replacement_chars = '
5      aaaaaaeeeeiiiiiooooouuuuAAAAAAEEEEIIIIIOOOOOUUUU'
6      translation_table = str.maketrans(accented_chars,
7      replacement_chars)
8      return text.translate(translation_table)

```

2. **Rimozione della punteggiatura e dei caratteri speciali:** sono stati rimossi i segni di punteggiatura e i caratteri speciali come < oppure >;

```

1  def clean_text(text):
2      text = remove_accents(text)
3      text = text.lower()
4      text = re.sub(r"[\.,;:!?'\-\"<> ]", "", text)
5      return text

```

3. **Trasformazione del testo in minuscolo;**

4. **Rimozione delle stopwords:** le stopwords sono parole comuni che appaiono frequentemente in un linguaggio ma che di solito non portano molto significato informativo nel contesto del documento. Esempi di stopwords in italiano includono parole come "a", "delle", "in", ecc. Queste parole sono utili per la struttura grammaticale ma generalmente non sono utili per l'analisi del contenuto;

```
1 def remove_stopwords(text):
2     ita_stopwords = stopwords.words('italian')
3     tokens = word_tokenize(text)
4     return [token for token in tokens if token not in
5             ita_stopwords]
```

5. **Lemmatizzazione:** la lemmatizzazione è il processo di riduzione delle parole flesse (cioè delle parole che hanno forme diverse a seconda del contesto grammaticale) alla loro forma base, detta **lemma**. Questo processo mira a ridurre le parole alla loro forma canonica, in modo che forme diverse della stessa parola possano essere trattate come lo stesso termine durante l'analisi.

```
1 def lemmatization(text):
2     text = ' '.join(text)
3     doc = nlp(text)
4     return " ".join([token.lemma_ for token in doc])
```

Una volta ottenuto il testo pre-processato, è stato possibile estrarre le keywords utilizzando KeyBERT.

```
1 def extract_keywords(text):
2     keywords_with_scores = kw_model.extract_keywords(text,
3     keyphrase_ngram_range=(1, 1))
4     return [keyword for keyword, score in keywords_with_scores]
5
6 # Pipeline di pre-processing del testo
7 def process_text(text):
8     text = clean_text(text)
9     text = remove_stopwords(text)
10    text = lemmatization(text)
11    return text
12
13 text_processed = text.apply(process_text)
14 keywords = text_processed.apply(extract_keywords)
```

Di seguito è riportato un esempio di keywords estratte:

	text	title	keywords
0	Semaforo verde alle vaccinazioni anti-Covid per i rifugiati che stanno arrivando in Italia dall'...	Ucraina, profughi in Italia: tamponi entro 48 ore dall'ingresso e vaccini	['vaccinare', 'vaccinazione', 'vaccinale', 'ucraina', 'vaccinato']
1	Risalgono i contagi in Italia, ma migliorano i dati dei ricoveri in terapie intensive e reparti ...	Coronavirus, risalgono i contagi: aumenta l'incidenza a 510 casi su 100mila abitanti. Terapie in...	['incidenza', 'ospedaliero', 'intensivo', 'sanitario', 'italia']
2	SE LA centrale nucleare ucraina di Zaporizhzhia, o quella di Chernobyl, dovessero essere violate...	Psicosi nucleare, i pericoli che le pillole di iodio non possono evitare - la Repubblica	['chernobyl', 'nucleare', 'iodio', 'radiazione', 'ucraina']
3	Social: Rss Facebook Twitter LinkedIn Youtube Instagram Edizioni Mediterraneo Europa-Ue NuovaEur...	Magonza-Dortmund - Sport - ANSA	['youtube', 'facebook', 'tweet', 'twitter', 'link']
4	Ucraina, diciassettesimo giorno di guerra. Potenti bombardamenti si sono registrati nella notte ...	Ucraina diretta. L'Ucraina accusa: ai soldati russi di Kharkiv l'ordine di sparare a civili e bi...	['ukrainska', 'ucraina', 'ucraini', 'ucraino', 'bombardamento']

Figura 1.4: Esempio output KeyBERT

1.3.5 Estrazione topic

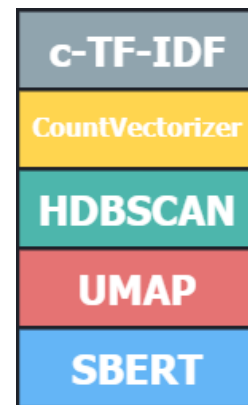
In questa fase si è proceduto all'estrazione dei topic a partire dal testo di tutte le notizie (corpus).

BERTopic

BERTopic è una libreria Python progettata per l'analisi dei temi basata su BERT, un modello avanzato di deep learning per il trattamento del linguaggio naturale. Questa libreria facilita l'estrazione automatica dei temi principali da grandi collezioni di documenti testuali senza la necessità di etichettare manualmente i dati.

Gli step seguiti sono stati i seguenti:

1. **Embedding**;
2. **Riduzione della dimensionalità**;
3. **Clustering gerarchico**;
4. **Topic Tokenization**;
5. **Weight tokens (c-TF-IDF)**.



1 - Embedding BERTopic prevede una trasformazione dell'input in una rappresentazione numerica. Nel progetto è stato utilizzato un embedder multilingua fornito dalla libreria SENTENCE-TRANSFORMERS.

```
1 # Step 1 Extract embeddings (SBERT)
2 embedding_model = SentenceTransformer('paraphrase-multilingual-
   MiniLM-L12-v2', device=device)
```

2 - Riduzione della dimensionalità Come impostazione di default, BERTopic utilizza l'algoritmo **UMAP (Uniform Manifold Approximation and Projection)** per eseguire la riduzione della dimensionalità.

UMAP (Uniform Manifold Approximation and Projection) è un algoritmo che preserva le relazioni di vicinanza tra i punti del dataset originale quando li proietta su uno spazio di dimensione inferiore. Utilizza l'ipotesi delle varietà per garantire che i punti vicini nel dataset originale siano mappati anche come punti vicini nello spazio ridotto. UMAP è flessibile, adatto a una vasta gamma di dati, e ottimizza la rappresentazione dei dati minimizzando una funzione di costo che tiene conto delle distanze tra i punti nei due spazi dimensionali.

I principali parametri configurati per UMAP sono stati:

- **nr_neighbors**: è il numero di punti campione vicini utilizzati durante l'approssimazione della varietà. Aumentare questo valore solitamente porta a una visione

più globale della struttura di embedding, mentre valori più piccoli portano a una visione più locale. Aumentare questo valore spesso porta alla creazione di cluster più grandi.

- **nr_components**: si riferisce alla dimensionalità degli embedding dopo la riduzione. Di default è impostato a 5 per ridurre la dimensionalità il più possibile pur cercando di massimizzare le informazioni mantenute negli embedding risultanti. Anche se ridurre o aumentare questo valore influenza la qualità degli embedding, il suo effetto è maggiore sulle prestazioni dell'algoritmo di clustering. Aumentare troppo questo valore potrebbe rendere difficile all'algoritmo clusterizzare gli embedding ad alta dimensionalità. Ridurre troppo questo valore potrebbe non fornire informazioni sufficienti negli embedding risultanti per creare cluster adeguati.

```
1 # Step 2 - Reduce dimensionality
2 umap_model = UMAP(n_neighbors = 15,
3                   n_components = 5,
4                   min_dist     = 0.0,
5                   random_state = random_state)
```

3 - Clustering gerarchico Per il clustering gerarchico si è utilizzato l'algoritmo **HDBSCAN** (Hierarchical Density-Based Spatial Clustering of Applications with Noise).

I parametri più rilevanti di questo algoritmo per il progetto sono:

- **min_cluster_size**: specifica la dimensione minima del cluster. Solo i cluster con un numero di punti maggiore o uguale al valore del parametro saranno considerati validi;
- **metric**: indica la metrica utilizzata per calcolare la similarità tra i punti.

```
1 # Step 3 - Clusterize reduced embeddings
2 hdbscan_model = HDBSCAN(min_cluster_size = min_cluster_size,
3                          metric='euclidean',
4                          cluster_selection_method='eom',
5                          prediction_data=True)
```

4 - Topic representations In BERTopic, le topic representations si riferiscono ai termini più rappresentativi o alle parole chiave che definiscono ciascun topic estratto dal modello. Queste rappresentazioni sono cruciali per comprendere e interpretare il contenuto di ciascun topic.

In particolare, CountVectorizer opera tokenizzando i dati testuali e contando le occorrenze di ciascun token. Successivamente, crea una matrice in cui le righe rappresentano i documenti e le colonne rappresentano i token. I valori delle celle indicano la frequenza di ciascun token in ogni documento. Questa matrice è conosciuta come *document-term matrix*.

```
1 # Step 4 - Tokenize topics
2 vectorizer_model = CountVectorizer(stop_words=ita_stopwords,
3                                   lowercase=False)
```

5 - c-TF-IDF Il c-TF-IDF (class-based Term Frequency-Inverse Document Frequency) è una variante del TF-IDF, un algoritmo che assegna un peso a ciascun termine che rappresenta l'importanza relativa di quel termine all'interno di un documento o un gruppo di documenti.

In BERTopic, per ottenere una rappresentazione accurata dei topic, il TF-IDF è stato adattato per funzionare a livello di cluster/topic anziché a livello di documento. Questa rappresentazione adattata del TF-IDF è chiamata c-TF-IDF e prende in considerazione ciò che rende i documenti in un cluster differenti dai documenti in un altro cluster.

```
1 # Step 5 - Create topic representation
2 ctfidf_model = ClassTfidfTransformer(reduce_frequent_words=True)
```

Pipeline complessiva di BERTopic Dopo aver istanziato le varie parti che compongono la pipeline di esecuzione di BERTopic, è stato possibile eseguire il modello.

```
1 topic_model = BERTopic(
2     min_topic_size = min_topic_size,
3     top_n_words = top_n_words,
4     embedding_model = embedding_model,           # Step 1 - Extract
5     umap_model = umap_model,                     # Step 2 - Reduce
6     hdbscan_model = hdbscan_model,               # Step 3 - Cluster
7     vectorizer_model = vectorizer_model,         # Step 4 - Tokenize
8     ctfidf_model = ctfidf_model,                 # Step 5 - Extract
9     nr_topics = nr_topics
10 )
```

E' stato fissato il numero esatto di topic da avere in uscita, pari a 25.

Un esempio di output è riportato in figura 1.5.

	text	title	topic	topic_name
0	Semaforo verde alle vaccinazioni anti-Covid per i rifugiati che stanno arrivando in Italia dall'...	Ucraina, profughi in Italia: tamponi entro 48 ore dall'ingresso e vaccini	3	3_paziente_vaccino_virus_covid
1	Risalgono i contagi in Italia, ma migliorano i dati dei ricoveri in terapie intensive e reparti ...	Coronavirus, risalgono i contagi: aumenta l'incidenza a 510 casi su 100mila abitanti. Terapie in...	-1	-1_anno_piu_essere_fare
2	SE LA centrale nucleare ucraina di Zaporizhzhia, o quella di Chernobyl, dovessero essere violate...	Psicosi nucleare, i pericoli che le pillole di iodio non possono evitare - la Repubblica	2	2_ucraino_russo_Russia_Putin
3	Social: Rss Facebook Twitter LinkedIn Youtube Instagram Edizioni Mediterraneo Europa-Ue NuovaEur...	Magonza-Dortmund - Sport - ANSA	18	18_00_primopiano_calcio_canale
4	Ucraina, diciassettesimo giorno di guerra. Potenti bombardamenti si sono registrati nella notte ...	Ucraina diretta. L'Ucraina accusa: ai soldati russi di Kharkiv l'ordine di sparare a civili e bi...	2	2_ucraino_russo_Russia_Putin

Figura 1.5: Esempio output BERTopic

Si nota dalla figura precedente la presenza del topic -1 , che raggruppa tutti gli articoli a cui l'algoritmo di clustering non è riuscito ad assegnare un topic specifico.

1.4 Preparazione dei dati per il RAG

Al momento i dati raccolti non sono adatti per essere utilizzati direttamente come base di conoscenza per il RAG. E' stato quindi necessario processare i dati salvati su MongoDB, prima dividendoli in modo opportuno (chunking), poi calcolando gli embedding ed infine salvandoli in un database vettoriale. Di seguito è riportata un'immagine che riassume il processo seguito.

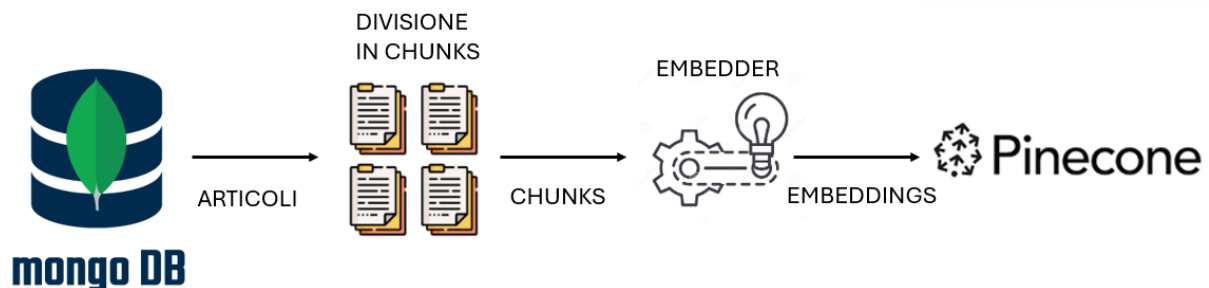


Figura 1.6: Pipeline preparazione dei dati per il RAG

1.4.1 MongoDB

Una volta raccolti i dati e pre-processati, è stato utilizzato MongoDB Atlas per la memorizzazione, il quale mette a disposizione un'istanza di MongoDB in cloud. Di seguito è riportato un esempio di documento.

```

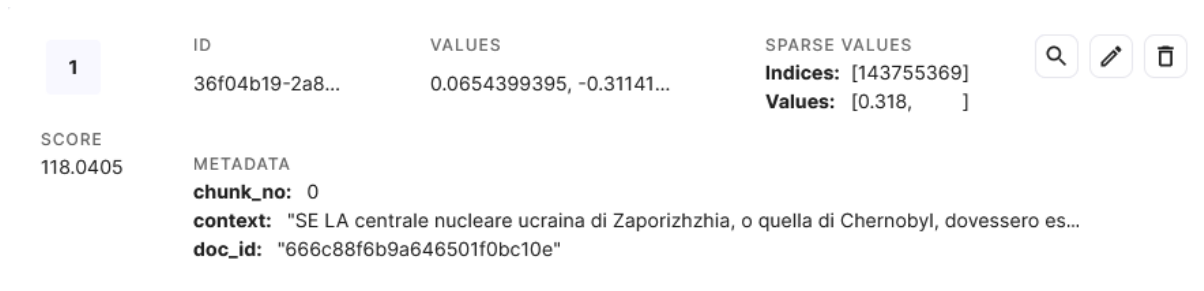
{
  "_id": ObjectId('666c88f6b9a646501f0bc10e'),
  "author": Array (2)
    0: "Antonella Donati"
    1: "Redazione Repubblica"
  "publish_date": "2022-03-11 14:19:05"
  "text": "SE LA centrale nucleare ucraina di Zaporizhzhia, o quella di Chernobyl..."
  "title": "Psicosi nucleare, i pericoli che le pillole di iodio non possono evita..."
  "url": "https://www.repubblica.it/salute/2022/03/10/news/psicosi_nucleare_i_pe..."
  "category": Array (1)
    0: "salute"
  "topic": 2
  "topic_name": "2_ucraino_russo_Russia_Putin"
  "keywords": Array (5)
    0: "chernobyl"
    1: "nucleare"
    2: "iodio"
    3: "radiazione"
    4: "ucraina"
  "publisher": "Repubblica"
}
  
```

Figura 1.7: Esempio documento MongoDB

E' stata creata un'unica collection *Articles*, in cui sono memorizzati tutti gli articoli recuperati.

1.4.2 Pinecone

Al fine di memorizzare gli embedding è stato utilizzato Pinecone, un database vettoriale. Tale tipologia di db è specializzata nella ricerca di dati di tipo vettoriale, garantendo efficienza e scalabilità. Di seguito è riportato un esempio di entry in Pinecone.






1	ID 36f04b19-2a8...	VALUES 0.0654399395, -0.31141...	SPARSE VALUES Indices: [143755369] Values: [0.318,]	  
SCORE 118.0405	METADATA chunk_no: 0 context: "SE LA centrale nucleare ucraina di Zaporizhzhia, o quella di Chernobyl, dovessero es... doc_id: "666c88f6b9a646501f0bc10e"			

Figura 1.8: Esempio entry Pinecone

Una possibile alternativa sarebbe stata memorizzare gli embedding direttamente su MongoDB Atlas, che permette la creazione di indici di ricerca sugli embeddings proprio come Pinecone. Tuttavia questa opzione è stata scartata poichè memorizzare anche gli embeddings su MongoDB avrebbe ridotto drasticamente il numero di articoli memorizzabili (dati i limiti del piano gratuito).

Si è deciso inoltre di memorizzare il testo relativo al singolo chunk anche nel db vettoriale. Questa scelta è motivata dal fatto di ridurre i tempi di latenza nella risposta del LLM, in quanto non ci sono overhead aggiuntivi relativi alla comunicazione tra MongoDB e Pinecone.

Chunking

La fase di chunking consiste nella suddivisione del testo in segmenti più piccoli (chunk). Questa fase risulta essere fondamentale prima di effettuare l'embedding e di memorizzarlo su Pinecone. La dimensione dei chunk avrà un enorme impatto sulle informazioni che verranno visualizzate durante la ricerca.

Con un chunk troppo grande abbiamo una perdita di specificità, in quanto quest'ultimo includerà una varietà di informazioni diverse, rendendo difficile per l'embedding catturare un tema specifico o un concetto chiave.

Con un chunk troppo piccolo si potrebbero avere vettori che non sono informativi o che non riescono a catturare completamente il significato del testo.

E' dunque necessario trovare un giusto trade-off nella scelta. Si è deciso pertanto di effettuare l'embedding su chunk creati splittando il testo originale dell'articolo ogni 300 token, dove un token corrisponde a circa una parola.

```
1 text_splitter = RecursiveCharacterTextSplitter(
2     chunk_size=300,
```



```
3     chunk_overlap=15,  
4     length_function=tokenizer_len,  
5     separators=["\n\n", "\n", " ", ""]  
6 )
```

Inizialmente si era ipotizzato di fare l'embedding sulle keywords. Tuttavia, le keywords rappresentano una versione condensata di un articolo, con proprietà differenti dal prompt inserito dall'utente (rimozione stopwords, punteggiatura, ecc.). Questo avrebbe portato ad alcune difficoltà nel recupero di articoli consistenti con il prompt inserito dall'utente. Pertanto si è deciso di non proseguire con questa strada. Inoltre, lo sparse embedding implementato successivamente riprende sostanzialmente questa idea, andandola a migliorare, attraverso l'algoritmo BM25.

Embedding dei chunk

Si è dunque proceduto a effettuare l'embedding dei singoli chunk. In particolare, è stato eseguito sia un dense che uno sparse embedding.

Dense Embedding Il dense embedding si riferisce a una rappresentazione in cui ogni parola o termine è mappato in uno spazio vettoriale di dimensione fissa, solitamente con un gran numero di dimensioni (ad esempio, centinaia o migliaia di dimensioni). In questa rappresentazione, ogni elemento del vettore contiene un valore numerico, che può essere un numero reale, positivo o negativo.

Questi vettori sono progettati in modo che parole con significati simili o che compaiono spesso in contesti simili siano rappresentate da vettori simili nello spazio vettoriale. Ad esempio, parole come "gatto" e "cane", che sono concetti simili nel contesto degli animali domestici, potrebbero avere vettori embedding simili in uno spazio vettoriale denso.

L'elaborazione di vettori densi può richiedere risorse computazionali significative a causa delle dimensioni elevate e della complessità della rappresentazione.

Nello specifico, è stato utilizzato un embedder della libreria SENTENCE-TRANSFORMER.

```
1 embedder = SentenceTransformer('nickprock/sentence-bert-base-italian  
    -xxl-uncased')
```

Sparse Embedding Lo Sparse embedding si riferisce a una rappresentazione in cui i vettori di embedding hanno molte dimensioni, ma la maggior parte dei valori in queste dimensioni è zero. In altre parole, il vettore di embedding per ogni parola ha pochi valori non nulli, mentre la maggior parte dei valori sono zero.

Questa rappresentazione è spesso utilizzata per gestire l'inefficienza dei vettori densi in termini di memoria e computazione. Ad esempio, in un vocabolario esteso con milioni di parole, la maggior parte delle parole è raramente usata in un dato contesto e pertanto avere un vettore denso per ogni parola sarebbe inefficiente in termini di risorse.

BM25 Per realizzare lo sparse embedding è stato utilizzato l'algoritmo BM25. Questo algoritmo tiene conto sia della frequenza di occorrenza di una parola sia di una normalizzazione della lunghezza di un documento per determinare la rilevanza di un documento per una specifica query. Questo algoritmo si basa su un'assunzione probabilistica, la quale assume che documenti rilevanti e non-rilevanti ai fini della risposta al prompt dell'utente seguono differenti distribuzioni statistiche.

Ci sono diversi elementi chiave in BM25 tra cui:

- **Term Frequency (TF)**: si riferisce al numero di volte in cui una parola compare in un documento.
- **Inverse Document Frequency (IDF)**: IDF misura l'importanza di una parola nell'intero corpus. Assegna un peso maggiore alle parole che sono rare e un peso minore a quelle che compaiono più spesso.
- **Document Length Normalization**: tale normalizzazione è necessaria per combattere l'impatto che ha la lunghezza del documento nella determinazione dei pesi delle parole. I documenti più lunghi hanno più occorrenze delle parole, introducendo un potenziale bias;
- **Query Term Saturation**: è un termine aggiuntivo per mitigare l'impatto di parole che compaiono molto spesso nei vari testi.

Questi 4 elementi chiave vengono poi considerati insieme per il calcolo di uno score $BM25(D, Q)$, dove D indica il documento e Q la query.

L'algoritmo è stato implementato nel progetto utilizzando il codice seguente.

```
1 sparse_embedder = BM25Encoder().default()
```

1.4.3 Sincronizzazione tra i due DB

Utilizzando due database distinti per i dati e gli embedding, è stato necessario implementare un meccanismo di sincronizzazione delle informazioni contenute nei due db. In particolare, all'aggiunta di un articolo su MongoDB deve corrispondere l'aggiunta su Pinecone degli embedding dei chunk associati a tale articolo. Inoltre, all'eliminazione di un articolo da MongoDB corrisponde la corrispondente eliminazione dei record relativi agli embedding dei chunk associati.

L'intero processo può essere osservato in Figure 1.9.

La nostra scelta è volta sulla creazione di un log che permettesse di ricondurre ad ogni id relativo agli articoli sul document Database l'id unico dei chunk associati sul vector Database, evitando quindi una serrata comunicazione tra i due database che avrebbe implicato numerose richieste per operazioni di lettura. Una qualsiasi operazione CRUD prevede quindi l'aggiornamento del log, il quale viene sempre controllato quando viene lanciata

l'operazione di aggiornamento del vector Database (sia per operazioni di creazione che di rimozione dei vettori).

Aggiornamento per id mancanti

L'aggiornamento del Vector Database parte con la definizione di una lista contenente tutti gli id presenti sul sul Document Database e che non siano contenuti nel log. Viene quindi eseguita una query su MongoDB laquale realizza una projection al fine di ottenere solo gli id dei documenti non ancora presenti sul VectorDB.

```
1 INFO:root:{ "timestamp": "2024-06-14T22:00:53.187262", "level": "
    VectorDB_update", "record_ID": "666c88f6b9a646501f0bc10c", "
    chunk_no": "0", "chunk_id": "36f04b14-2a88-11ef-a267-08bfb81a3431
    " }
```

Listing 1.1: Esempio di entry del log

Logica di gestione dei batch

L'elaborazione dei chunk, dei vettori di embedding ed il loro caricamento sul vector database viene eseguito in batch. Si va quindi a definire una dimensione del batch (batch_limit) per numero di records (i.e. articoli) in cui la scelta della dimensione del batch è lasciata all'utente. Viene definito quindi un loop che eseguirà un numero di cicli pari a numero_records/batch_limit. In ogni ciclo viene effettuata una query richiedendo una porzione dei records limitata dal batch_limit, per poi passare al processo di chunking del testo e inserendo nei metadati del vettore sia l'id del documento che il numero rappresentativo del chunk. Successivamente viene generato un id unico basato sul timestamp (che diverrà poi l'id del record sul Vector DB). Infine si ha creazione dei vettori di embedding (sparse e dense) ed ogni vettore viene quindi aggiunto ad una lista che sarà passata all'operazione di upsert.

Al termine di ogni batch viene eseguito, infine, l'aggiornamento del log. Nella versione Serverless di Pinecone le operazioni sui vettori sono eseguite in modo sequenziale. Tuttavia, in altri tipi di client offerti dal servizio, è possibile effettuare l'operazione di upsert in modo asincrono velocizzando ulteriormente l'operazione.

Infine, è riportato in figura 1.9 uno schema finale che riassume il processo di creazione del dataset, caricamento dei dati nei db e la comunicazione tra questi ultimi.

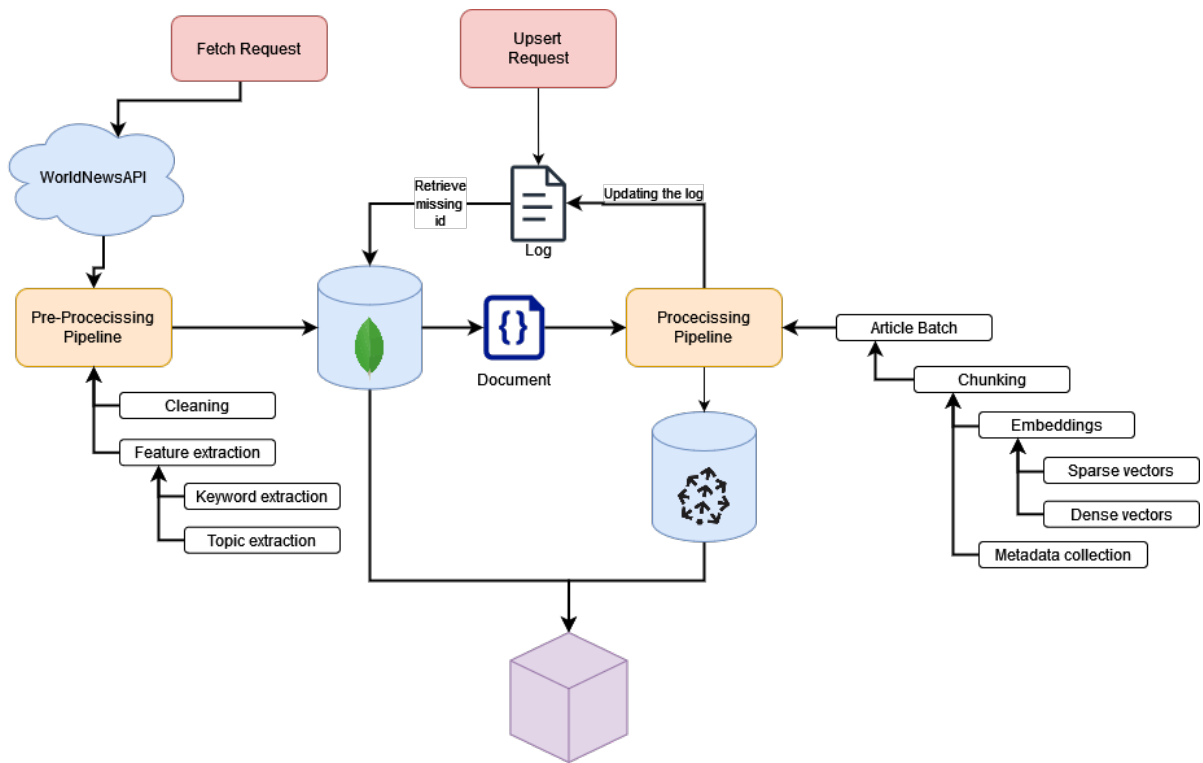


Figura 1.9: Flusso dei dati riguardante la comunicazione e sincronizzazione dei database

Capitolo 2

Chatbot

L'obiettivo è quello di realizzare un Chatbot in grado di rispondere a domande sulla veridicità o meno di notizie inserite in input dall'utente. Si è fatto uso di LLM e di una tecnica che permette di estendere la loro conoscenza.

2.1 Pipeline complessiva

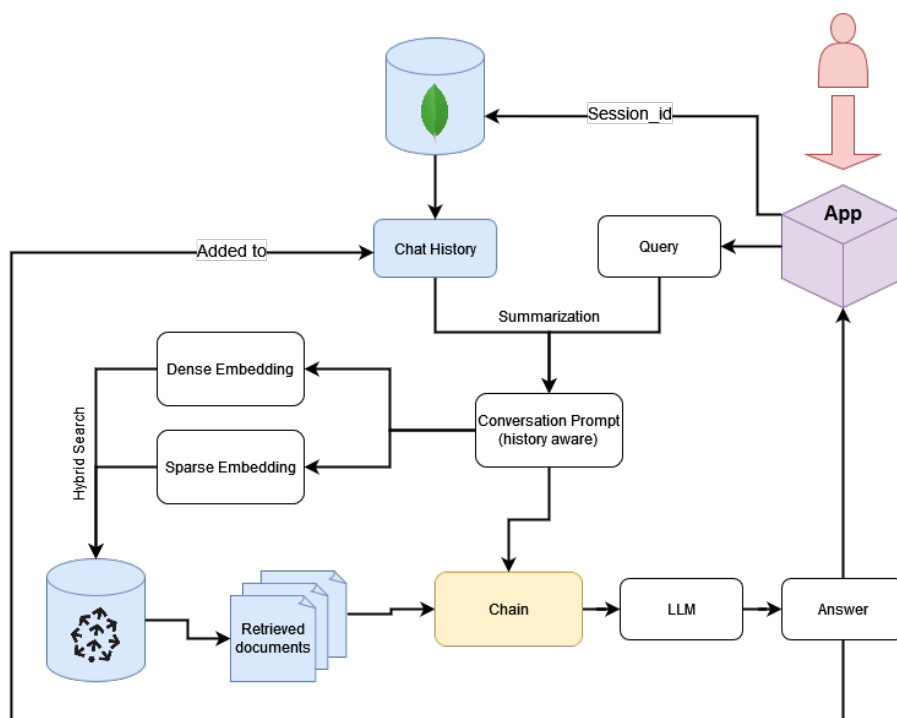


Figura 2.1: Schema funzionamento Chatbot

2.1.1 LLM

Il modello di LLM selezionato per un applicativo di RAG può essere cruciale nella performance riguardo due aspetti fondamentali: la velocità di elaborazione della richiesta e

la generazione di una risposta coerente e pertinente ma che contemporaneamente rispetti il contesto della discussione. In particolare, l'ultimo aspetto sottolineato richiede l'addestramento del modello su dati di qualità e che coprano molti ambiti diversi. Tuttavia, non tutti i modelli open-source possono godere di un addestramento di questo tipo. Nonostante oggi esistano piattaforme che permettono l'utilizzo di modelli open-source (ad esempio HuggingFace), tali modelli, seppur performanti, possono essere messi in difficoltà o richiedere di essere calibrati (eventualmente implementando anche un fine-tuning) in base all'applicazione che si intende creare.

Per questo applicativo il LLM scelto è "gpt-3.5-turbo", uno dei modelli proprietari di OpenAI. Il motivi alla base di questa scelta sono:

- **Contesto esteso:** ChatGPT-3.5 può utilizzare una finestra di contesto fino a 16,385 token. La grandezza di tale contesto è ben bilanciata in accordo al numero di token medio dei chunk di testo da noi definito. Una finestra di tale dimensioni permette di tener conto allo stesso tempo sia della discussione tenutasi tra user e AI sia dell'inserimento delle informazioni recuperate dal retriever, mantenendo coerenza e la pertinenza delle risposte.
- **Adattabilità:** supponendo che il modello abbia avuto accesso in fase di addestramento ad una grande mole di dati di qualità ci si aspetta che, a differenza dei modelli open-source, abbia la capacità di far riferimento ad informazioni importanti e settorizzate. Un aspetto cruciale dell'applicativo è il fatto che l'utente può chiedere un approfondimento su un gran numero argomenti che coprono settori molto diversi tra loro.

La scelta di rivolgersi ad un modello proprietario che possa essere chiamato attraverso un API è dovuta anche all'esigenza dell'applicazione di ottenere rapidamente una risposta dal modello. Una lunga attesa nell'ottenere la risposta avrebbe un peso negativo determinante nella user experience. Questa motivazione è stata determinante nella scelta del modello LLM implementato: inizialmente l'applicativo prevedeva infatti l'implementazione del modello *Llama3* e la sua esecuzione su un hardware in locale. Tuttavia il tempo di attesa medio per la generazione di una risposta era di 20 secondi, superando quindi quello che è stato ritenuto un tempo accettabile nella generazione della risposta. Tuttavia non è possibile esprimersi in merito alle performance del modello, che avranno sicuramente risentito dell'hardware molto limitato.

2.1.2 RAG

Per estendere la conoscenza del LLM, è stata utilizzata la tecnica RAG che ne permette un'estensione in maniera "dinamica" senza necessità di eseguire fine-tuning espliciti. Sebbene si tratti di una scelta obbligata in questo contesto, in quanto effettuare fine-tuning sarebbe impossibile dal punto di vista computazionale, è in realtà anche la scelta

migliore: tale tecnica, infatti, nel caso in cui si volessero effettuare degli aggiornamenti ad elevata periodicità, richiederebbe solo l'inserimento dei nuovi documenti all'interno del database vettoriale.

In linea generale, il funzionamento del RAG prevede due fasi, articolate in tre punti:

1. L'user sottomette una domanda, la quale viene trasformata in embedding;
2. l'embedding viene utilizzato per effettuare una ricerca di similarità tra gli embedding dei chunk all'interno del database vettoriale. Dai record più simili verranno estratti i testi (fase di **retrieve**);
3. tali testi sono poi "iniettati", sotto forma di contesto, all'interno del prompt del LLM (fase di **generation**).

Retrieve

La Hybrid search è un tipo di ricerca che può fornire un contributo cruciale nelle applicazioni basate su RAG, garantendo risposte accurate dai modelli di linguaggio di grandi dimensioni (LLM). Sebbene la ricerca semantica possa permetta di ottenere buoni risultati con domande simili a quelle umane, potrebbe avere difficoltà a catturare le parole chiave. In questo approccio quindi viene usata in combinazione con la ricerca tradizionale basata su parole chiave per migliorare i risultati.

La hybrid search è stata implementata attraverso la classe *PineconeHybridSearch* messa a disposizione dal framework Langchain, con la quale verrà istanziato il retriever.

La domanda dell'utente viene passata al retriever, dopodiché si esegue l'encoding (dense e sparse) della domanda ed i vettori prodotti vengono forniti in input ad una funzione di scaling *hybrid_convex_scale()* che eseguirà uno scaling di questi vettori tramite un parametro $\alpha \in [0, 1]$. Tale funzione permette di stabilire se attribuire un peso maggiore ad uno dei due encoding, permettendo anche di usare solo uno dei due (i.e. con $\alpha = 1$ verrà usato solo il vettore con dense encoding). Si ottengono quindi i seguenti vettori:

$$scaled_dense = dense \cdot \alpha$$

$$scaled_sparse = sparse \cdot (1 - \alpha)$$

Tali vettori verranno utilizzati per inviare una query al vector database.

Generation

Nella fase di generation, dopo aver individuato gli articoli più attinenti al prompt dell'utente, viene generata attraverso LLM la risposta finale del Chatbot. Tale risposta tiene

conto non solo della domanda fornita dall'utente ma anche della conversazione tenutasi fino a quel momento. Per ottenere questo risultato è necessario implementare la "memoria".

Si può immaginare la conversazione di un chatbot come uno scambio di oggetti "messaggio" inviati dall'user e dall'AI. Questi messaggi possono essere inseriti quindi in dei prompt template messi a disposizione da Langchain. Il generico question answering prompt assume quindi la forma:

1. ("system", "Sei un assistente per un question answering tasks...").
2. MessagesPlaceholder("chat_history").
3. ("human", "input")

Tuttavia questa soluzione potrebbe diventare dispendiosa in termini sia economici che di computazione (i.e. alcuni modelli proprietari scalano il credito in base alla lunghezza dei token del prompt). Va inoltre considerato che alcuni modelli possiedono una ristretta finestra di contesto in input e quindi passare attraverso il prompt una lunga conversazione sarebbe impraticabile. Infine si deve sottolineare anche che a seguito di alcune ricerche si è dimostrato che testi "eccessivamente" lunghi possono impattare le performance e le capacità del LLM.

Per fronteggiare questo problema quindi è stata implementata una funzionalità che esegue un riassunto della chat history e quest'ultima viene usata al suo posto (i.e. come contesto). Il nuovo prompt composto da riassunto della chat history e domanda dell'utente è quindi utilizzato dal retriever e dal LLM.

Attraverso il meccanismo delle chain messo a disposizione da Langchain quindi generiamo un messaggio per il LLM a cui contemporaneamente vengono passati:

- la domanda dell'utente;
- la chat history tenutasi in questo momento;
- i documenti rilevanti alla conversazione, recuperati dal retriever.

Osservazioni sulla chat history La generazione di una domanda basandosi sulla totalità della conversazione tenute precedentemente è un processo che richiede l'implementazione di una persistenza nella chat: i messaggi inviati dall'utente e dal rispettiva risposta del LLM vengono quindi conservati in modo che un utente possa, una volta essersi disconnesso dal servizio, avere la capacità di tornare alla precedente sessione.

I messaggi vengono salvati come singoli record in una collezione dedicata sul document database. Le chat history di uno specifico utente e conversazione sono identificate attraverso un id unico che viene generato all'apertura di una nuova finestra di conversazione

all'interno dell'applicazione. La scelta di un sistema di messaggistica persistente ha quindi permesso di raggiungere due risultati: migliorare il contesto fornito al LLM e aggiungere una nuova funzionalità per l'user.

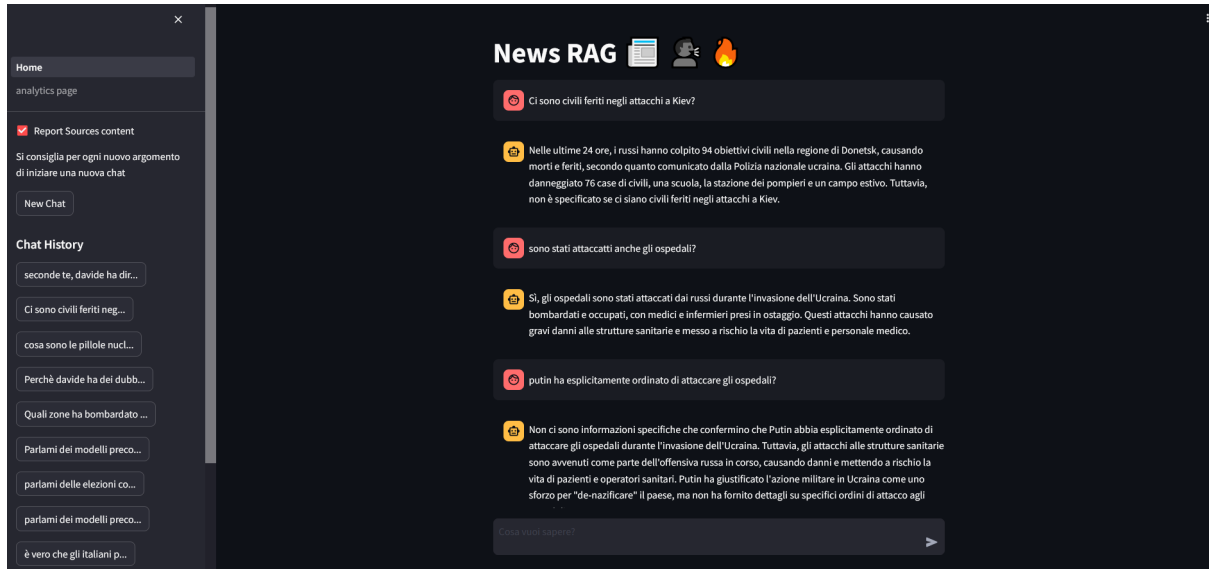


Figura 2.2: Chat dashboard

Capitolo 3

Analytics

Una pagina dell'applicazione streamlit è dedicata alla data analytics, che permette all'utente di esplorare gli articoli raccolti filtrando per vari parametri, tra cui:

- `publish_date`: data di pubblicazione dell'articolo, è possibile scegliere una finestra temporale;
- `author`: autori dell'articolo;
- `category`: categoria dell'articolo estratta dall'url;
- `keywords`: parole chiavi rappresentanti l'articolo (estratte con Keybert).

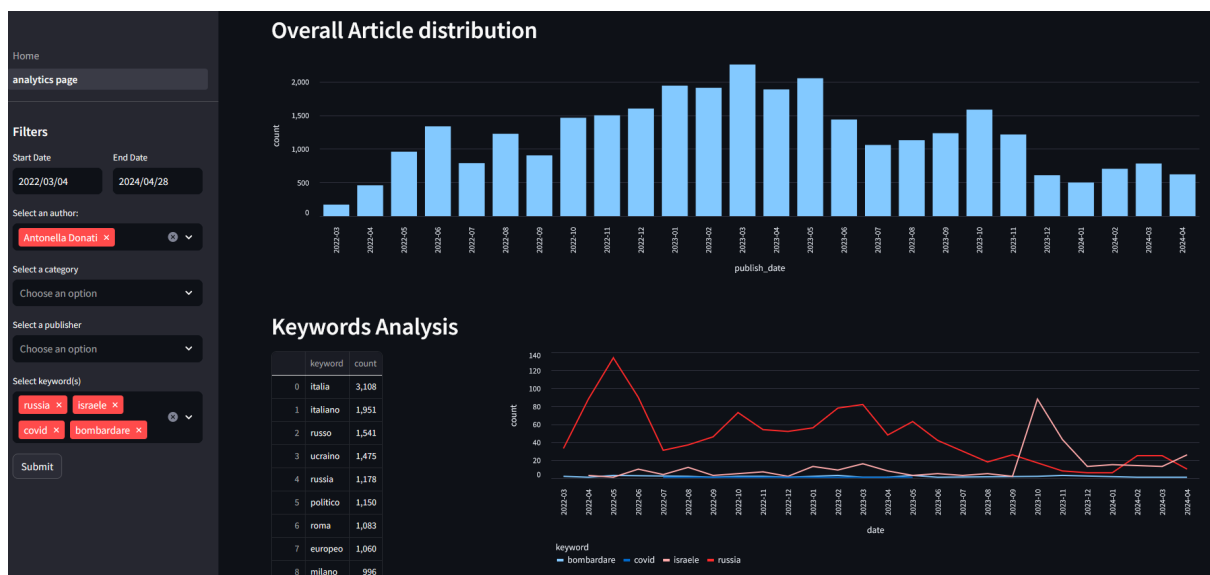


Figura 3.1: Overview della pagina di analytics

3.1 Analytic 1: Distribuzione degli articoli nel tempo

Per avere un'idea generale sulla quantità di articoli scritti nel tempo, è stato inserito un barchart che mostra il conteggio degli articoli per ogni mese. I dati da mostrare sono stati ottenuti attraverso una query di aggregazione su mongoDB.

```

1 pipeline = [
2     {
3         "$addField": {
4             "parsed_date": { "$dateFromString": { "dateString": "
5                 $publish_date", "format": "%Y-%m-%d %H:%M:%S" } }
6         },
7     },
8     {
9         "$group": {
10             "_id": {
11                 "year": { "$year": "$parsed_date" },
12                 "month": { "$month": "$parsed_date" }
13             },
14             "count": { "$sum": 1 }
15         },
16     },
17     {
18         "$sort": { "_id.year": 1, "_id.month": 1 }
19     },
20     {
21         "$project": {
22             "_id": 0,
23             "publish_date": {
24                 "$concat": [
25                     { "$toString": "$_id.year" },
26                     "-",
27                     { "$cond": { "if": { "$lt": ["$_id.month", 10] }, "then": "0", "else": "" } },
28                     { "$toString": "$_id.month" }
29                 ]
30             },
31             "count": 1
32         }
33 ]
34
35 results = list(collection.aggregate(pipeline))
36 # Separate lists for unique dates and counts
37 unique_dates = [result['publish_date'] for result in results]
38 counts = [result['count'] for result in results]
39 df_tmp = pd.DataFrame(results)
40 st.write("## Overall Article distribution")
41 st.bar_chart(data=df_tmp, x="publish_date", y="count", color=None,
42             width=None, height=None, use_container_width=True)

```

3.2 Analytic 2: Analisi delle keywords

Questa analytic mira a restituire il conteggio nel tempo degli articoli aventi le keywords selezionate. A sinistra è presente un dataframe che mostra le top 10 keyword presenti negli articoli, mentre a destra il linechart delle keywords selezionate dall'utente.

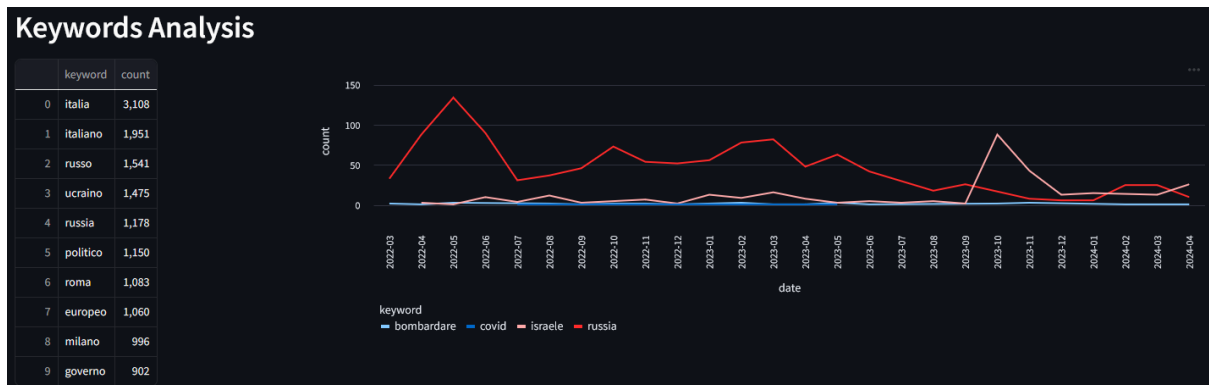


Figura 3.2: Keywords analysis

Dalla figura 3.2 si può notare come sia possibile per l'utente selezionare varie keywords contemporaneamente e confrontarne l'andamento nel tempo.

Di seguito le pipeline implementate per ottenere le due rappresentazioni:

```

1 st.write("## Keywords Analysis")
2 coll, col2 = st.columns([1, 3])
3 start_date_tmp = str(start_date)+" T00:00:00"
4 end_date_tmp = str(end_date)+" T23:59:59"
5 pipeline_keywords = [
6     {
7         "$match": {
8             "publish_date": {
9                 "$gte": start_date_tmp,
10                "$lt": end_date_tmp
11            }
12        },
13    },
14    {
15        "$unwind": "$keywords"
16    },
17    {
18        "$group": {
19            "_id": "$keywords",
20            "count": { "$sum": 1 }
21        }
22    },
23    {
24        "$sort": { "count": -1 }
25    }
26 ]
27 df_tmp = pd.DataFrame(list(collection.aggregate(pipeline_keywords)))

```

```

28 df_tmp.rename(columns={'_id':'keyword', 'count':'count'}, inplace=
    True)
29 coll.dataframe(df_tmp.head(10))
30 # keywords chosen by user
31 if keywords_to_count:
32     pipeline = [
33         {
34             "$match": {
35                 "keywords": { "$in": keywords_to_count }
36             },
37         },
38         {
39             "$addFields": {
40                 "publish_month": {
41                     "$substr": ["$publish_date", 0, 7] # Extract
42                     the "YYYY-MM" part of the publish_date
43                 }
44             },
45         },
46         {
47             "$unwind": "$keywords"
48         },
49         {
50             "$match": {
51                 "keywords": { "$in": keywords_to_count }
52             },
53         },
54         {
55             "$group": {
56                 "_id": {
57                     "month": "$publish_month",
58                     "keyword": "$keywords"
59                 },
60                 "count": { "$sum": 1 }
61             },
62         },
63         {
64             "$sort": { "_id.month": 1, "_id.keyword": 1 } # Sort by
65             month and then by keyword
66         }
67     ]
68     # Execute the aggregation
69     results = list(collection.aggregate(pipeline))
70     reshaped_results = [
71         {
72             "date": result['_id']['month'],
73             "keyword": result['_id']['keyword'],
74             "count": result['count']
75         }
76         for result in results
77     ]
78     df_tmp = pd.DataFrame(reshaped_results)
79     col2.line_chart(data=df_tmp, x="date", y="count", color="keyword")

```



```

2 df = load_df(start_date, end_date, author, category, publisher)
3 if author:
4     st.write("# Author Section")
5     # viz 1: distribution of articles w/ categories
6     df_mod = df
7     df_mod['publish_date'] = df_mod['publish_date'].apply(
8         cut_date)
9     counts = []
10    for date in year_list:
11        count = df_mod[df_mod['publish_date'] == date].shape[0]
12        counts.append(count)
13    df_tmp=pd.DataFrame(zip(year_list, counts), columns=["Date",
14        "counts"])
15    st.write("### {author} articles distribution".format(author=
16        author[0]))
17    st.bar_chart(data=df_tmp, x="Date", y="counts")
18    # viz 2
19    auth_coll1, auth_col2 = st.columns(2)
20    df_mod['category'] = ["unclassified" if x==[] else x[0] for
21        x in list(df_mod['category'])]
22    plot2 = sns.countplot (x= df_mod["publisher"],hue=df_mod['
23        category'])
24    bio_template = "{author}    un giornalista ... nuove culture
25        ".format(author=author[0])
26    stringa_contatti = "[X] (www.x.com)\n[IG] (www.instagram.com)"
27    auth_coll1.write("#### Biography:"+" \n")
28    auth_coll1.write(bio_template)
29    auth_coll1.write("Social: "+stringa_contatti)
30    auth_col2.write("#### At a glance")
31    auth_col2.pyplot(plot2.get_figure(), clear_figure =True)
32    st.write("#### {author} articles repository".format(author=
33        author[0]))
34    st.dataframe(df_mod[['publish_date','title','url']])
35
36    # viz 3: wordcloud
37    corpus = ' '.join(df_mod['text'])
38    corpus = process_text(corpus)
39    corpus = ' '.join(corpus)
40    wc = WordCloud(background_color="white", max_words=30, width
41        =1000, height=500)
42    wc_image = wc.generate(corpus)
43    plt.imshow(wc_image, interpolation="bilinear")
44    plt.axis("off")
45    plt.show()
46    st.pyplot()

```

3.4 Analytic 4: Visualizzazione modello BERTopic

Questa sezione è dedicata allo studio del modello ottenuto da BERTopic attraverso varie funzioni di visualizzazione. Da notare che **le funzioni mostrate in seguito sono state**

eseguite nel notebook *Topic_BERTopic*. Possiamo avere un quadro generale del modello attraverso la funzione *topic_model.get_topic_info()*:

	Topic	Count	Name	Representation	Representative_Docs
0	-1	12424	-1_anno_piu_essere_fare	[anno, piu, essere, fare, euro, altro, potere, ...]	[prospettiva crescita area euro deteriorato al...
1	0	4622	0_squadra_gara_gol_finale	[squadra, gara, gol, finale, partita, vincere, ...]	[paese vai evento trovare luogo proponga Festi...
2	1	4125	1_pd_governo_melone_partito	[pd, governo, melone, partito, ministro, Itali...	[senato russo commemorazione berlusconi aula m...
3	2	2966	2_ucraino_russo_Russia_Putin	[ucraino, russo, Russia, Putin, Kiev, guerra, ...]	[ucraina diciassettesimo giorno guerra potente...
4	3	696	3_paziente_vaccino_virus_covid	[paziente, vaccino, virus, covid, tumore, infe...	[chiamare tecovirimat momento farmaco riporre ...]
5	4	661	4_incidente_morire_ferito_morto	[incidente, morire, ferito, morto, ospedale, c...	[orsa amarena essere uccidere ieri sera fucila...
6	5	652	5_nord_sud_caldo_pioggia	[nord, sud, caldo, pioggia, temperatura, tempo...	[sbalzo termico crisi primavera appena iniziar...
7	6	530	6_elettrico_litro_prezzo_gas	[elettrico, litro, prezzo, gas, auto, benzina, ...]	[girona Mazda continuare battere ferro motore ...]
8	7	503	7_film_regista_attore_cinema	[film, regista, attore, cinema, Oscar, the, pr...	[mancare solo coda sirena sedurre sembrare ina...
9	8	500	8_Papa_papa_Francesco_vaticano	[Papa, papa, Francesco, vaticano, cardinale, c...	[citta vaticano – Papa Francesco dovere rinunc...
10	9	417	9_arresto_denaro_giudice_Messina	[arresto, denaro, giudice, Messina, carcere, m...	[magistrato Procura Palermo carabinieri ros in...
11	10	339	10_google_intelligenza_apple_digitale	[google, intelligenza, apple, digitale, artifi...	[seattle motore ricerca oggi piu frenetico sfi...
12	11	334	11_volo_migrante_aereo_ita	[volo, migrante, aereo, ita, passeggero, nave, ...]	[tanto viaggio speranza trasformare nuovo trag...
13	12	296	12_scolastico_studente_scuola_docente	[scolastico, studente, scuola, docente, aborto...	[fare finire oppure no compito assegnare estat...
14	13	288	13_cinese_taiwan_Cina_Pechino	[cinese, taiwan, Cina, Pechino, xi, Corea, pec...	[aereo nave guerra cinese attraversare linea m...
15	14	283	14_tasso_inflazione_rialzo_bce	[tasso, inflazione, rialzo, bce, calo, prezzo, ...]	[cio contare economia distribuzione reddito fr...
16	15	275	15_cane_gatto_animale_cucciolo	[cane, gatto, animale, cucciolo, proprietario, ...]	[poco rumore straziante piangere cane diverso ...]
17	16	252	16_trump_donald_repubblicano_tycoon	[trump, donald, repubblicano, tycoon, Biden, Y...	[oggi giorno partito politico potere arrestare...
18	17	189	17_principe_Carlo_harry_regina	[principe, Carlo, harry, regina, camilla, Re, ...]	[chiudere pubblico camera ardente bara elisabe...
19	18	144	18_00_primopiano_calcio_canale	[00, primopiano, calcio, canale, abbonato, vid...	[social rss Facebook Twitter linkedin youtube ...]
20	19	117	19_cookie_anso_it_tracciamento	[cookie, anso, it, tracciamento, Ansa, canale, ...]	[scegliere accettare cookie diprofazione tra...
21	20	115	20_Musk_twitter_elon_tesla	[Musk, twitter, elon, tesla, account, utente, ...]	[nuovo grana elon Musk nuovo pagina saga twitt...
22	21	112	21_turismo_turistico_vacanza_destinazione	[turismo, turistico, vacanza, destinazione, tu...	[Italia turismo business cima classifica mondi...
23	22	95	22_sciopero_sindacato_trasporto_proclamare	[sciopero, sindacato, trasporto, proclamare, u...	[confederazione unitario base (cub) Slai cob...
24	23	88	23_colore_look_abito_make	[colore, look, abito, make, moda, indossare, c...	[quando essere dare annuncio presenza Balencia...
25	24	85	24_spaziale_luna_galassia_solare	[spaziale, luna, galassia, solare, orbita, tel...	[quando formare anello saturno risposta riport...

Figura 3.5: Topic info

3.4.1 Topics per documento

Attraverso la funzione *topic_model.visualize_documents(text_processed_series, embeddings=embeddings)* vengono mostrati gli embedding degli articoli in un piano 2D, dove ogni punto è colorato in base al topic di appartenenza.

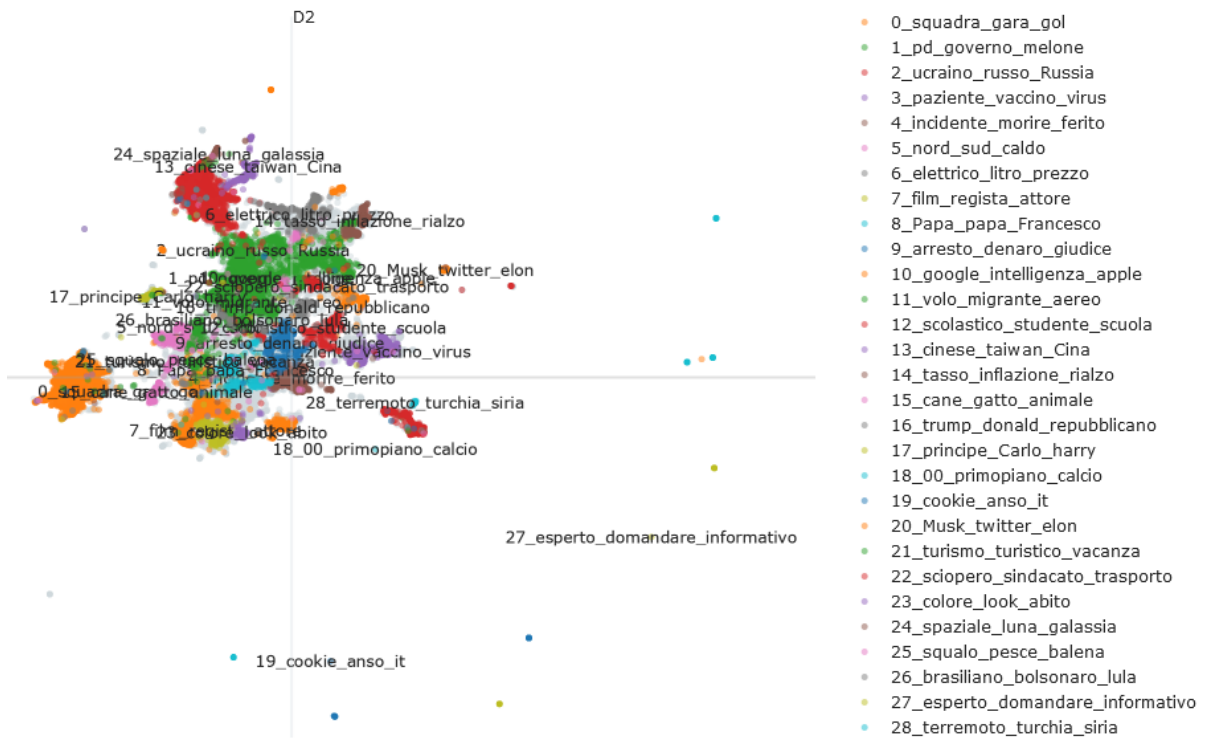


Figura 3.6: Topics per documento

3.4.2 Word Scores per topic

Attraverso la funzione `topic_model.visualize_barchart()` viene effettuato un barchart per i top 8 topic, in cui vengono rappresentati i termini descrittivi del topic ed il loro c-TF-IDF score.

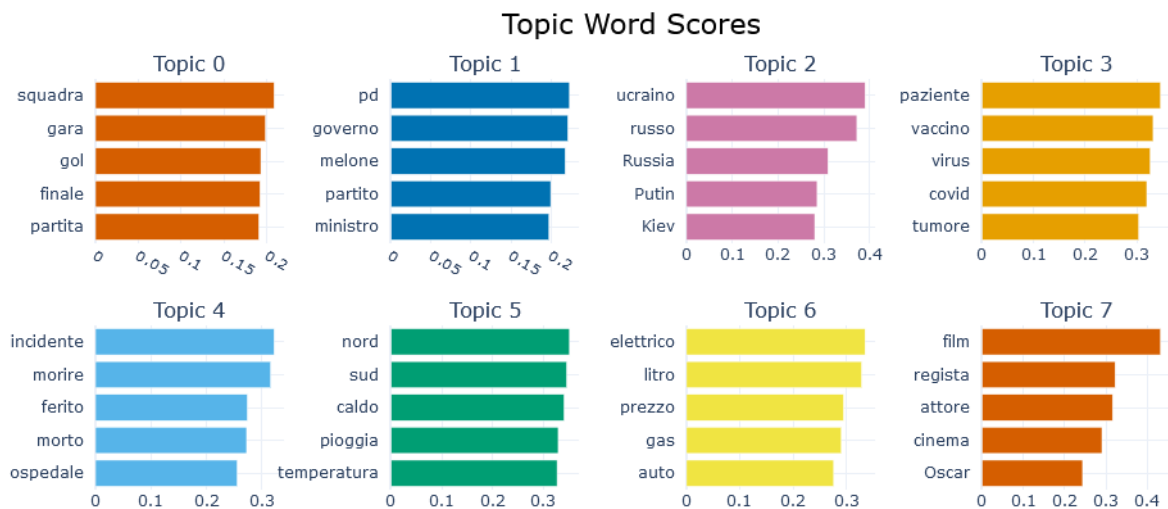


Figura 3.7: Topic Word Scores

Capitolo 4

Sviluppi Futuri

In questo capitolo sono proposte alcune idee per migliorare e sviluppare ulteriormente l'attuale versione del progetto.

4.1 Gestione di più utenti

Ad ogni conversazione effettuata con il LLM è associato un ID univoco, chiamato *Sessione_ID*, il quale è associato a sua volta ad un unico utente. Un possibile sviluppo futuro sarebbe quello di associare ogni conversazione ad uno specifico utente, e aggiungere una nuova collection USER, la quale tiene traccia, per ogni utente, delle *Sessione_ID* ad esso associate.

4.2 NER e Graph RAG

In futuro, si propone di arricchire il progetto implementando una Named Entity Recognition (NER) integrata con un sistema di Graph Retrieval-Augmented Generation (Graph RAG). Questa evoluzione permetterà di migliorare la precisione e l'efficacia del processo di verifica delle notizie, consentendo una gestione più avanzata e strutturata delle informazioni. In particolare:

- **NER:** l'integrazione della NER consentirà di identificare automaticamente entità rilevanti come testate, autori, luoghi, date e altri elementi chiave all'interno dei testi delle notizie. Questo processo aiuterà a isolare e confrontare con maggiore precisione le informazioni rilevanti nei diversi articoli di giornale presenti nel database;
- **Graph RAG:** il Graph RAG utilizza grafi di conoscenza per rappresentare e navigare tra le relazioni tra le diverse entità riconosciute dalla NER. Integrando la NER con il Graph RAG, sarà possibile generare risposte più accurate e contestualizzate, migliorando la capacità del sistema di rilevare incongruenze e verificare la veridicità delle affermazioni in modo più efficiente.

4.3 Biografia autore

Nella versione attuale del progetto, la biografia dell'autore inserita è un place-holder. Un possibile sviluppo futuro potrebbe essere quello di avere una biografia specifica per ogni autore, così come i link per i suoi contatti social. Questo implicherebbe la creazione di una nuova collection all'interno di MongoDB e la futura possibilità di collegare gli articoli all'autore ma anche a blog posts o tweets del giornalista permettendo così all'utente di poter esplorare argomenti e temi affrontati da un giornalista in modo completo e facendo sì che questo diventi un riferimento per la sua informazione in merito ai temi ricercati.