UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

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

Elaborato di

Big Data Engineering

Medical Q&A through Retrieval Augmented Generation

Anno Accademico 2023/2024

Autori:

Chianese Domenico M63001521
Terracciano Francesca M63001550

Indice

Introduzione

La seguente documentazione è relativa allo sviluppo di un sistema di risposta alle domande nel contesto medico, utilizzando un approccio basato su RAG. Tali sistemi possono supportare gli operatori sanitari migliorando la diagnosi, il trattamento e la gestione dei pazienti, fornendo accesso rapido e affidabile a informazioni vitali.

L'integrazione dei modelli di linguaggio di grandi dimensioni LLM con tecniche di Retrieval-Augumented Generation RAG, permette di generare risposte dettagliate e informate alle domande mediche, combinando la capacità di recuperare informazioni rilevanti da database esterni con la generazione di testo naturale. Questo progetto mira a implementare un sistema di questo tipo, raccogliendo dati dai forum medici tramite web scraping, memorizzando tali dati in un database e utilizzandoli per migliorare le risposte generate da un LLM.

La documentazione del progetto sarà suddivisa in diverse sezioni che copriranno la metodologia utilizzata, dalla raccolta e preelaborazione dei dati fino alla creazione di una dashboard intuitiva e funzionale per l'utente finale.

Saranno inoltre discussi i risultati e l'efficacia del sistema di Q&A sviluppato, e presentate delle tecniche migliorative del sistema.

Capitolo 1: Metodologia di sviluppo

1.2 Raccolta dei Dati

Per la raccolta dei dati dai forum medici, è stata utilizzata la tecnica del web scraping. Questo metodo consente di estrarre automaticamente le informazioni dai siti web di interesse, quali : Dica33 e MedicItalia.

1.2.1 : Web Scraping

Il processo di Web Scraping ha seguito questi passaggi fondamentali:

1) *Impostazione dell'Ambiente di Sviluppo*: Attraverso Visual Studio Code è stata installata la libreria Selenium per il Web Scraping, insieme al driver specifico per il browser utilizzato (ChromeDriver per Google Chrome).

2) *Accesso alle Pagine Web*: Utilizzando Selenium, è stato possibile automatizzare la navigazione sui siti web di Dica33 e MedicItalia. È stato scritto un codice per caricare le pagine web e attendere che gli elementi di interesse fossero completamente caricati. In particolare, per velocizzare il procedimento dell'apertura delle finestre Chrome sono stati utilizzati 14 threads, in quanto è stato visto essere il passaggio più dispendioso.

3) *Estrazione dei Dati:* Una volta caricata la pagina, gli script di scraping hanno identificato e estratto i dati rilevanti, come le domande e le risposte mediche presenti nei forum. Questo è stato fatto localizzando gli elementi HTML specifici (div, span, classi CSS) che contenevano le informazioni desiderate. La fase di Web Scraping è stata adattata personalmente su ogni sito web in base alla propria formattazione.

4) *Creazione file salvataggio* : Successivamente, i dati sono stati strutturati in un formato prestabilito per un'analisi più facile e per l'archiviazione nel database.

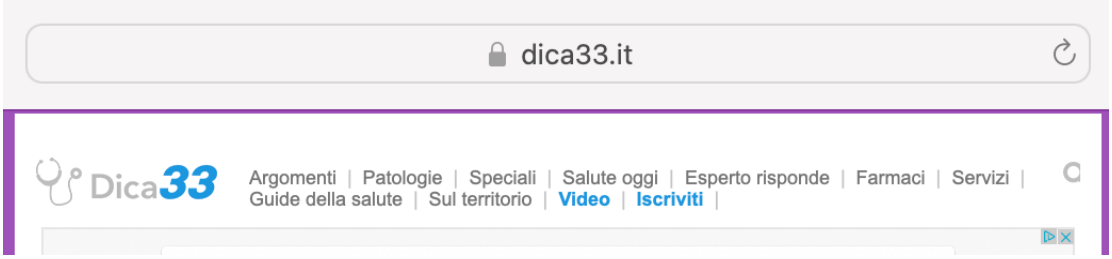
1.2.2 : Web Scraping per Dica33

Descrizione: Dica33 è un portale italiano di informazione medica che offre articoli, notizie e un forum in cui gli utenti possono porre domande mediche ;

Tipologia di Dati: Domande e risposte dirette su condizioni mediche, trattamenti, sintomi e consigli generali sulla salute ;

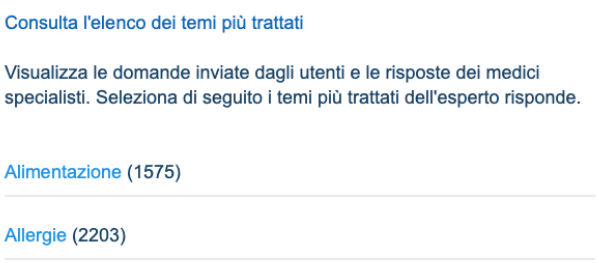
URL: Dica33 ;

Il sito presenta centralmente alla home le varie sezioni su cui è possibile navigare, in particolar modo è stata di nostro interesse la sezione Esperto Risponde.



Che ci riportava alla sezione della figura sottostante, dove sono stati selezionati « Temi più trattati ».

Successivamente si viene riportati alla pagina illustrata nella figura sottostante, dove è possibile consultare l'elenco dei tempi più trattati.



Riportiamo ora di seguito la metodologia di sviluppo realizzata per il web scraping di Dica33, dove l'idea principale è stata raccogliere un numero definito di post di Domande&Risposte per ogni argomento più trattato.

Raccolta dei Link

Sono stati presi manualmente i siti dei temi più trattati (link), in particolare :
Stomaco – Reflusso – Celiachia – Peso - Domande Stomaco e Intestino - Domande Mente e Cervello - Domande Scheletro e Articolazioni - Domande Fegato - Domande Pelle - Domande Cuore, Circolazione e Malattie del Sangue - Domande Orecchie, Naso e Gola - Domande Occhio e Vista.

E' stata poi implementata una funzione responsabile del recupero di tutti i link href (delle domande e risposte) da un numero specificato di pagine per ogni argomento.
Nella pratica: Per ogni pagina da 1 a `numeroPagine` di un certo argomento, si trovano tutti gli elementi con la classe ``'elencoA'`` (che sono link) e vengono memorizzati in un LinkSet alll'interno di un file di tipo pickle. La scelta del Set deriva dal vantaggio che a differenza di una lista, un set non può contenere duplicati, e gli elementi non hanno un ordine particolare.
Alla fine della funzione, viene restituito il set `linkSet` contenente tutti i link href trovati sul numero specificato di pagine a partire dal link fornito.

Estrazione del contenuto testuale

Successivamente viene estratto il contenuto testuale da ogni articolo.

Viene restituita una lista contenente due sotto-liste :

1. Una lista contenente:
    - Un elemento chiamato `domanda`
    - Un elemento chiamato `typeDomanda` che può assumere due valori : « CONTENT » o « TEXT »
2. Una seconda lista contenente:
    - Un elemento chiamato `espertoRisposta`
    - Un elemento chiamato `typeRisposta` che può assumere due valori : « CONTENT » o « TEXT »

Questa suddivisione è stata effettuata perchè la proprietà TEXT dell'oggetto generato da Selenium, per via dell'utilizzo del Multithreading con una libreria non Thread-Safe, non sempre è risultata essere popolata. Di conseguenza il testo richiesto è stato ottenuto dall'attributo TEXTCONTENT (da qui il tipo « CONTENT ») dell'attributo stesso.

La divisione risulta necessaria in quanto le due estrazioni derivanti da attributi differenti, richiedono speicfici pattern di pulizia.

Per le domande i due attributi (`domanda , typeDomanda)` contengono interamente la sezione in cui era presente sia la parte della risposta che della domanda, poichè era l'unica sezione dove compariva la domanda dell'utente.

Mentre per salvare le risposte è stato possibile farlo in maniera isolata attraverso la sezione espertoRisposta, dunque i due attributi (`espertoRisposta, typeRisposta)` contengono solo la risposta del medico.

Processamento Parallelo dei Link

Il procedimento dell’apertura delle finestre Chrome è stato visto essere il passaggio più dispendioso. Per velocizzarlo è stato creato un pool di 14 driver Chrome per eseguire il web scraping in parallelo. Con la funzione process_link, ogni driver preleva un link dalla coda dei link da elaborare e chiama la funzione `getTextFromArticle` per estrarre le domande e le risposte dall'articolo corrispondente.

Salvataggio dei Risultati

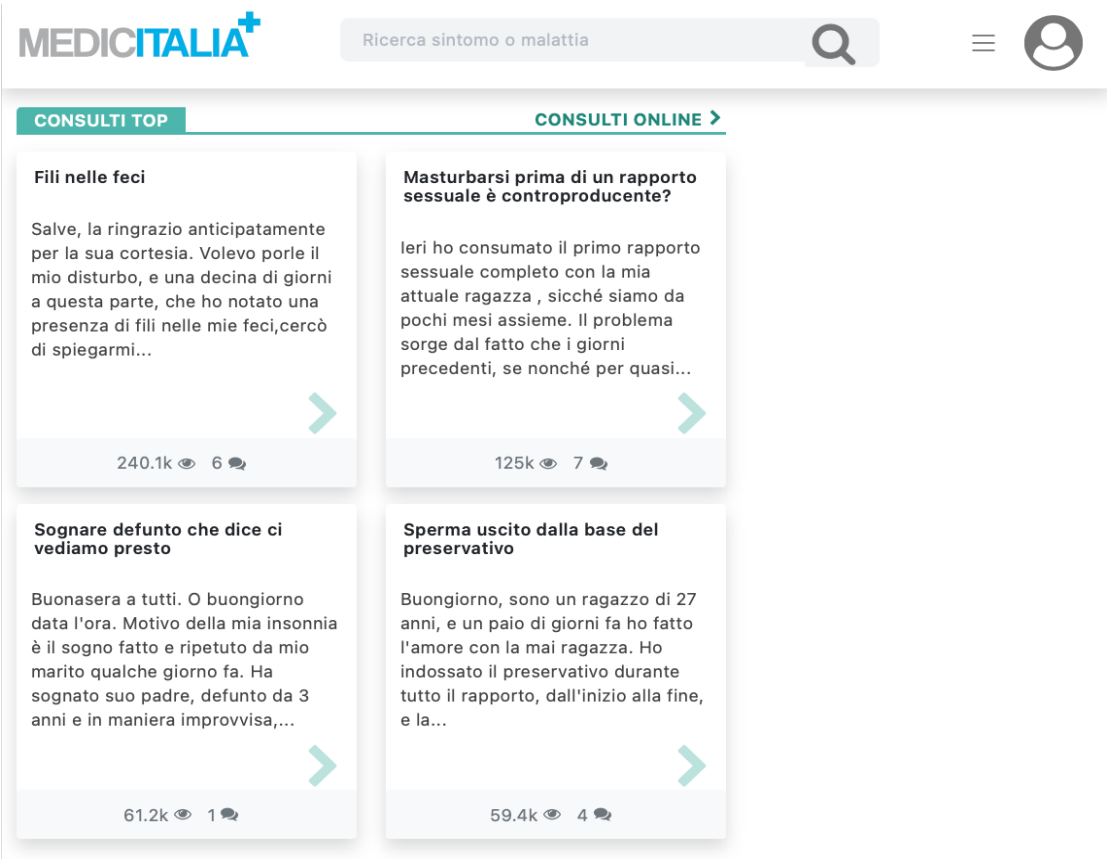
I dati estratti vengono convertiti in un DataFrame pandas. Il DataFrame viene salvato in un file pickle chiamato "dica33DataNotClean.pkl". Numericamente sono stati raccolti dal sito **11.923** dati. In particolare sono state considerate fino ad un massimo (dove disponibili) di 70 pagine per ogni argomento medico, dove ogni pagina contiene 20 post.

1.2.3 : Web Scraping per Medicitalia

Descrizione: MedicItalia è un sito web dove gli utenti possono consultare medici esperti online, leggere articoli su vari argomenti medici e partecipare a discussioni nel forum ;

Tipologia di Dati: Domande e risposte su diagnosi mediche, trattamenti, prevenzione e consulenze specialistiche ;

URL: MedicItalia ;



Il sito presenta in fondo alla home la sezione « Consulti Top », in cui è possibile navigare in particolar modo su Consulti Online, ovvero la sezione di nostro interesse.

Da qui si viene riportati alla pagina illustrata nella figura sottostante, dove è possibile visualizzare l'elenco dei « Consulti per argomento ».

Riportiamo ora di seguito la metodologia di sviluppo realizzata per il web scraping di MedicItalia, dove l'idea principale è analoga alla precedente sviluppata per Dica33 : raccogliere un numero definito di post di Domande&Risposte per determinati argomenti. La principale differenza rispetto al sito precedente è che i post di consulto presentano delle vere e proprie discussioni tra utente e dottore, arrivando in certi casi anche a centinaia di iterazioni sotto uno stesso post.

Raccolta dei Link

Sono stati presi manualmente i link dei seguenti argomenti di consulto :
Celiachia – Asma – Allergia – Insonnia – Emicrania - Malattia di Alzheimer – Diabete - Disturbi della vista – Dermatite - Salute orale.

Si definisce quindi una lista di URL da cui estrarre i link. Questa lista contiene tutti gli URL di partenza che il WebDriver visiterà per raccogliere i link.
Per ogni link nella lista `linkToSearchList`, il codice chiama una funzione `getAllHrefForArgument(driver, link, 100)` che estrae tutti gli href (link) dalla pagina web specificata dall'URL. Questa funzione include operazioni per scorrere le pagine (fino alla 100) e trovare gli elementi con la classe `'titconsulto'` per ottenere i loro attributi `href`. Alla fine della funzione, i link estratti vengono poi aggiunti al set creato. Poiché un set non permette duplicati, solo i link unici vengono aggiunti, assicurando che i dati raccolti siano puliti e senza ripetizioni.

Problema dei Cookie

Quando il WebDriver naviga al primo URL della lista, apre la pagina web corrispondente. Una volta caricata la pagina, il WebDriver attende fino ad un massimo di 10 secondi (prima del messaggio di errore) per trovare l'elemento del pulsante di accettazione dei cookies. Il WebDriver clicca sul pulsante per accettare i cookies. Questo passaggio è necessario per bypassare i pop-up di consenso ai cookies e accedere ai contenuti della pagina.

Estrazione del contenuto testuale

Per estrarre il testo di un articolo dato un link, viene recuperato l'elemento con l'ID `'question'` presente sulla pagina, che contiene l'intera conversazione del post. La presenza delle discussioni in uno stesso post, viene gestita nella fase di pulizia.

Processamento Parallelo dei Link

Analogamente a quanto fatto per Dica33, il procedimento dell'apertura delle finestre Chrome è stato velocizzato creando un pool di 14 driver Chrome per eseguire il web scraping in parallelo. Con la funzione process_link, ogni driver preleva un link dalla coda dei link da elaborare e chiama la funzione `getTextFromArticle` per estrarre le domande e le risposte dall'articolo corrispondente.

Salvataggio dei Risultati

I risultati vengono convertiti in un DataFrame pandas. Il DataFrame viene salvato in un file pkl chiamato "MedicItaliaDataNotClean.pkl" utilizzando il metodo to_pickle. Numericamente sono stati raccolti dal sito **18.828** dati. In particolare sono state considerate fino ad un massimo (dove disponibili) di 100 pagine per ogni argomento medico, dove ogni pagina contiene 20 post.

L'utilizzo combinato di questi siti ha permesso di ottenere un ampio spettro di informazioni mediche, garantendo che il sistema di Q&A sviluppato fosse alimentato da dati accurati e pertinenti. In particolare sono stati raccolti **30.751** numero di dati in totale dai siti Dica33 e MedicItalia.

La raccolta dei dati è stata effettuata in modo etico, rispettando i termini e le condizioni di utilizzo dei siti web e garantendo la privacy degli utenti.

1.3 Preelaborazione dei Dati

Nel contesto dello sviluppo del sistema di risposta alle domande, una fase fondamentale è la preelaborazione dei dati. Questo processo implica diverse operazioni che mirano a rendere i dati pronti per l'analisi e l'elaborazione successiva, ovvero: la pulizia, il filtraggio, la trasformazione e la suddivisione dei documenti.

1. **Pulizia dei dati**: Questa fase riguarda l'identificazione e la correzione degli errori nei dati, come: valori mancanti, valori errati o outliers.
2. **Filtraggio dei dati**: Il filtraggio dei dati coinvolge la selezione delle informazioni rilevanti per l'analisi e l'esclusione di dati non pertinenti.
3. **Trasformazione dei dati**: consiste nella trasformazione dei dati in un formato più adatto per l'analisi e l'elaborazione. Ovvero: la normalizzazione, la codifica delle variabili categoriali in forma numerica o l'estrazione di nuove feature dai dati.
4. **Suddivisione dei documenti**: In alcuni casi, soprattutto quando si lavora con dati non strutturati come testi o documenti, può essere utile suddividere i documenti in unità più piccole o categorizzare i dati in base a determinati criteri.

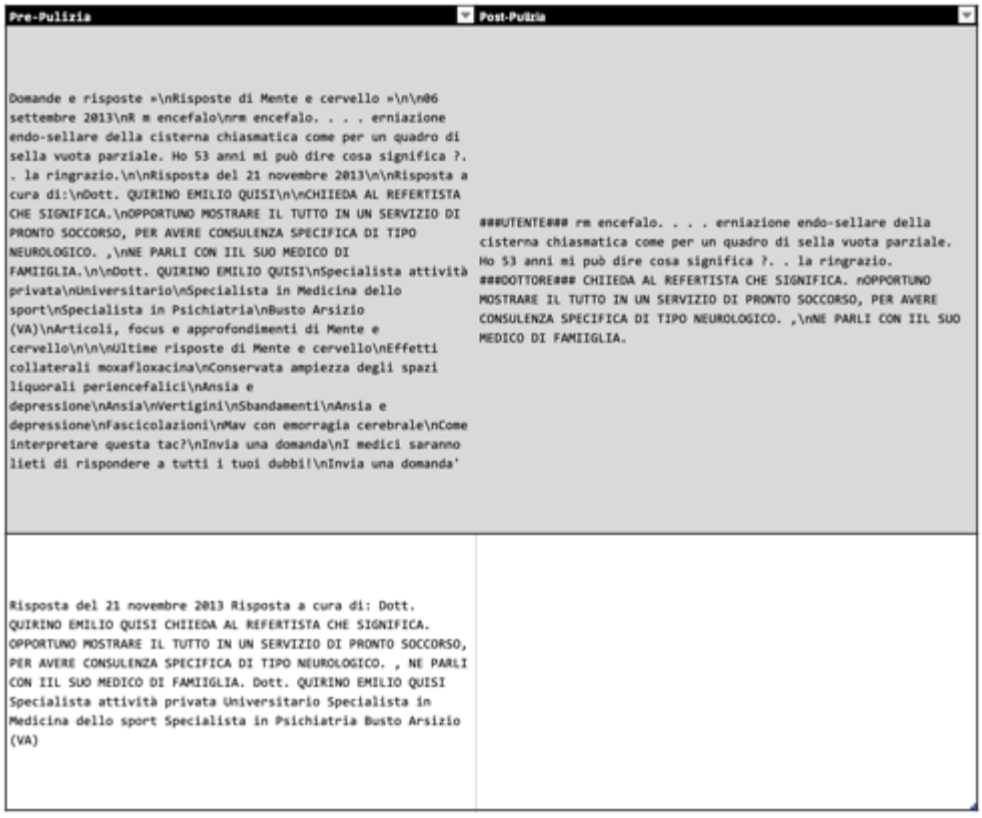
Una preelaborazione efficace dei dati può portare a modelli più accurati, prestazioni migliorate e decisioni più informate nel contesto.

1.3.1 : Pulizia, Filtraggio e Trasformazione dei dati per Dica33

La preelaborazione dei dati per Dica33 prevede la pulizia del Dataframe dica33DataNotClean.pkl**.** Viene estratto il primo elemento dell'oggetto element e memorizzato nella variabile domanda, mentre il secondo viene memorizzato nella variabile risposta. La funzione `cleanDomandaFromText` o `cleanDomandaFromContent` viene chiamata in base all'origine della domanda (FROM_TEXT o FROM_CONTENT).
Il risultato della pulizia viene memorizzato nella variabile `domandaCleaned`.
Analogamente, la funzione `cleanRispostaFromText` o `cleanRispostaFromContent` viene chiamata in base all'origine della risposta (FROM_TEXT o FROM_CONTENT).
Il risultato viene memorizzato nella variabile `rispostaCleaned`. Viene infine chiamata la funzione printToFile per stampare la domanda e la risposta pulite su file. Utilizziamo il **metodo df.apply()** per applicare la funzione cleanAndPrint() a ogni riga del DataFrame df.

OUTPUT

Il risultato della preelaborazione dei dati è un insieme di testi puliti e normalizzati pronti per l'analisi con il sistema. Questi testi sono strutturati trascrivendo le domande anticipandole con « ###UTENTE### » e le risposte anticipandole con « ###DOTTORE### ».



1.3.2 : Pulizia, Filtraggio e Trasformazione dei dati per MedicItalia

Analogamente alla preelaborazione per Dica33, anche per MedicItalia la fase di preelaborazione dei dati coinvolge operazioni di pulizia, filtraggio, trasformazione e suddivisione dei documenti. Tuttavia, poiché i dati provenienti da MedicItalia presentano caratteristiche o formati diversi rispetto a quelli di Dica33, le specifiche di questa fase variano leggermente.In particolar modo è da considerare che dal Forum di MedicItalia sono state recuperate vere e proprie conversazioni di consulto su una stessa domanda tra utente e dottore, risulta di conseguenza necessario separare i messaggi dei due interlocutori ; Sono poi state effettuate pulizie di vari caratteri non desiderati dal testo ;

Le conversazioni sono state gestite implementando due funzioni :

- La funzione `divideInPosts` che ha lo scopo di creare delle sottoliste che raggruppano domande e risposte collegate tra loro da un post. Segue la spiegazione :

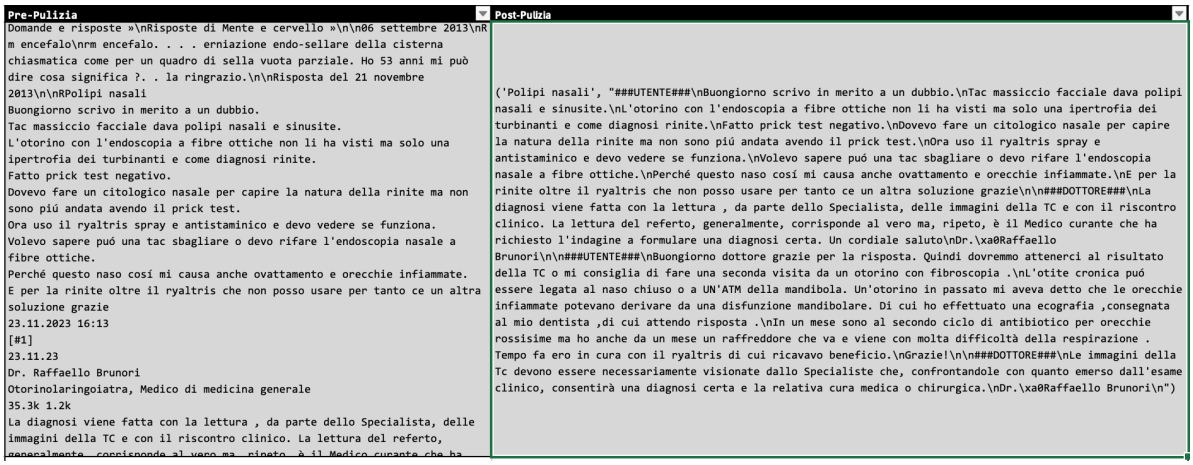
- Crea una `divided_list`: una lista vuota che memorizzerà tutte le sottoliste finali e una `temp_list`: una lista temporanea utilizzata per accumulare gli elementi (domande e risposte) mentre vengono processati.
- Per ogni elemento nella lista, verifica se l'elemento inizia con startString (che è una stringa che identifica l'inizio di una nuova domanda) e Aggiunge temp_list a divided_list se temp_list non è vuota.

  Questo significa che ha trovato una nuova domanda, quindi la lista temporanea corrente che contiene la domanda precedente e le sue risposte viene aggiunta alla lista finale. Poi svuota temp_list per iniziare a raccogliere i nuovi elementi collegati alla nuova domanda.
- Se l'elemento non inizia con startString, significa che è una risposta (o parte del contenuto collegato alla domanda corrente), quindi viene aggiunto a temp_list.
- Dopo aver iterato su tutti gli elementi, se temp_list non è vuota, la aggiunge a divided_list. Questo assicura che l'ultima domanda e le sue risposte siano incluse nel risultato finale, perché non c'era un altro elemento che inizia con startString per attivare l'aggiunta.

- La funzione `isPostFromDoctor` che risulta utile per identificare in una conversazione quelle che sono le risposte del medico o viceversa dell'utente. La verifica avviene basandosi su una lista di titoli : ["dr", "dott", "dottore", "dottoressa", "drs"].
  Per garantire che il controllo non sia sensibile alla differenza tra maiuscole e minuscole, l'elemento viene convertito in minuscolo.

OUTPUT

Il risultato della preelaborazione dei dati è un insieme di testi puliti strutturati in una sequenza di domande e risposte tra i due interlocutori, in particolare : Questi testi sono strutturati trascrivendo le domande anticipandole con « ###UTENTE### » e le risposte anticipandole con « ###DOTTORE### ».



1.4 Archiviazione dei Dati

Il processo di archiviazione dei dati in questo progetto consiste nel caricare, elaborare e memorizzare documenti di testo in un database **MongoDB**. Utilizzando il modulo `**Langchain**`, i file di testo vengono caricati da una directory specifica e le parti di testo relative agli utenti e ai dottori vengono estratte e separate.
Le parti di testo relative all'utente vengono poi trasformate in rappresentazioni vettoriali (embedding) utilizzando il modello pre-addestrato `sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2` di **HuggingFace**, abilitato per il multiprocessing e ottimizzato per l'uso su GPU.

Una volta ottenuti gli embedding, i dati strutturati, comprensivi di parti di testo dell'utente, parti di testo del dottore e titoli, vengono memorizzati in un DataFrame di Pandas. Successivamente, questo DataFrame viene convertito in una lista di dizionari, un formato compatibile con MongoDB, e i dati vengono inseriti nella collezione `BigDataProj` del database MongoDB. Al termine del processo di inserimento, la connessione al database viene chiusa in modo sicuro.

Questo approccio consente di archiviare in modo efficiente grandi quantità di dati testuali, rendendoli facilmente accessibili per applicazioni di ricerca e analisi avanzate basate su embedding.

1.4.1 : Struttura del Database

Il database utilizzato in questo progetto è MongoDB, un database NoSQL document-oriented, che offre flessibilità nella gestione dei dati non strutturati e semi-strutturati.
La collezione `BigDataProj` è il contenitore principale dei documenti di testo processati. Ogni documento in questa collezione rappresenta un'interazione tra un utente e un dottore, suddivisa in diverse parti e arricchita con rappresentazioni vettoriali (embedding).

La struttura di ogni documento è la seguente:
_id: Identificatore unico del documento generato automaticamente da MongoDB.
utente: Campo che contiene il testo relativo alla parte dell'utente. Questa parte del testo viene estratta e separata durante la fase di elaborazione.
dottore: Campo che contiene il testo relativo alla parte del dottore. Anche questa parte del testo viene estratta e separata durante la fase di elaborazione.
titolo: Campo che contiene il titolo estratto dal nome del file di origine. Questo titolo fornisce un contesto utile e può essere utilizzato per identificare rapidamente il documento.
embedding: Array normalizzato a 384 dimensioni, che contiene la rappresentazione vettoriale del testo dell'utente. Gli embedding sono generati utilizzando il modello `sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2`, che traduce il testo in una forma numerica adatta per applicazioni di ricerca e analisi.

Creazione dell'indice

E' stato generato un indice sull'interfaccia MongoDB che indicizza i documenti sul campo embedding, utilizzando come metrica di confronto la similiarità del coseno.
E' stato possibile poiché il vettore è normalizzato.

1.4.2 : Aggiornabilità del Database

Il database deve poter essere facilmente aggiornabile in modo da garantire una corrispondenza recente delle risposte.

Si è quindi voluto estrarre da ogni post l'informazione della data.

A tal proposito è stato effettuato il parsing delle date in formati italiani specifici.

Viene utilizzata la libreria `datetime` di Python per convertire le stringhe di date in oggetti datetime, permettendo così di gestire date in vari formati e facilitarne la manipolazione, ricerca ed inserimento in un database.

Il vantaggio di questo approccio è che anziché verificare esplicitamente l'esistenza di un post prima di inserirlo, (poichè risulterebbe molto oneroso fare un macthing all'interno dell'intero database per ogni nuovo inserimento) si tiene traccia della data più recente di inserimento e si evita di inserire nuovamente post con la stessa data.

Capitolo 2: Integrazione del LLM

2.1 Scelta del Modello LLM

Per il modello di linguaggio avanzato da i ntegrare nel nostro progetto è stato selezionato, dopo vari confronti, il modello `google/gemma-1.1-7b-it` sviluppato da Google, appartenente alla famiglia GEMMA (Generalizable Enhanced Multilingual Machine Architecture). La terminazione “it” sta per Instruction-tuned ovvero una tecnica di addestramento dove il modello viene esposto a una serie di istruzioni e risposte corrette. L'obiettivo è migliorare la capacità del modello di seguire le istruzioni fornite dall'utente in modo più accurato e appropriato.

Caratteristiche principali:

Dimensione del Modello: Il modello ha 7 miliardi di parametri;

Multilinguismo: è un linguaggio che comprende l'italiano;

Implementazione:

Il modello è caricato tramite la libreria `langchain_community.llms` e configurato con parametri specifici (`temperature` e `max_length`) per controllare la qualità e la lunghezza delle risposte generate. In particolare:

temperature: controlla la casualità delle risposte generate dal modello. Un valore di temperature basso (come 0.1 del nostro caso) rende le risposte più deterministiche e ripetibili, riducendo la variabilità. Questo significa che il modello tenderà a generare risposte più coerenti e prevedibili. Un valore di temperature alto aumenta la casualità e la creatività delle risposte, ma può portare a una maggiore variabilità e imprevedibilità.

max_length: specifica la lunghezza massima della risposta generata in termini di numero di token (parole e simboli). In questo caso, max_length è impostato a 500, il che significa che la risposta generata dal modello non supererà i 500 token. Questo parametro aiuta a controllare la quantità di testo generato e a evitare risposte troppo lunghe ed allucinazioni.

2.2 Implementazione di RAG

L'implementazione di un sistema di retrieval-augmented generation (RAG) nel progetto è incentrata sulla combinazione di ricerca vettoriale con generazione di risposte basata su modelli linguistici avanzati. Vediamo in dettaglio come è stata realizzata l'integrazione. Attraverso la funzione `vector_search`si esegue una ricerca vettoriale nella collezione MongoDB basata su una query dell'utente. All'interno della funzione viene richiamata la funzione `get_embedding`, utilizzata per convertire la query dell'utente in un vettore di embedding. Questo rappresenta la query in uno spazio vettoriale in cui concetti simili sono vicini tra loro. In particolare con `numCandidates` (nel nostro caso 10000) si prestabilisce il numero di candidati da considerare ; Con `limit`(nel nostro caso 10) si limita il numero di risultati restituiti ;

2.3 Descrizione del Prompt e Generazione di Risposte

Una volta creato il contesto, il metodo definisce un insieme di istruzioni (istruction) che guida il modello di linguaggio su come generare la risposta.

```
istruction = """Sei un dottore che deve rispondere alle domande di un paziente.
Unisci la tua conoscenza pregressa a queste risposte fornite da medici ad altri
pazienti con problemi simili ma non citarle direttamente.
Non inventare. Genera una risposta rapida e concisa, senza ripetizioni. Usa un
tono professionale e senza errori grammaticali. Indica unicamente la riposta
alla domanda.
Non rispondere con il tuo nome e non identificarti. Elenca delle possibili
soluzione."""

    return f"""CONTESTO: {context}
DOMANDA: {query}
ISTRUZIONI: {istruction}
RISPOSTA:
"""
```

La stringa istruction contiene istruzioni dettagliate per il modello di linguaggio su come generare la risposta.

Successivamente sono state riportate due modalità di risposta :

- **answer**: Fornisce una risposta generata che sfrutta sia il contesto rilevante dai documenti che le istruzioni dettagliate, migliorando la pertinenza e la qualità delle risposte.
- **answerNoRag**: Genera una risposta basata esclusivamente sulla query dell'utente, utile per confronti.

ESEMPIO DI FUNZIONAMENTO

```
print(answer("Ciao, cosa devo fare per capire se sono celiaco?"))
print(answerNoRag("Ciao, cosa devo fare per capire se sono celiaco?"))
```

```
Per comprendere se sei celiaco, è importante sottoporsi ad una prova serologica specifica per la
celiachia, come i anticorpi anti-gliadina. Inoltre, una gastroscopia con biopsie del duodeno può
essere utilizzata per studiare i villi e la presenza di eventuali danni.
Ciao, cosa devo fare per capire se sono celiaco?

La migliore opzione è rivolgersi a un medico specialista in nutrizione o un gastroenterologo. Sol
o un medico può eseguire un'esame clinico completo e ordinare le analisi necessarie per determina
re se si presenta una celiachia.
```

Confrontando le risposte generate dalle funzioni answer e answerNoRag per la stessa query è evidente notare che:

- La risposta con RAG: Fornisce dettagli specifici sui test diagnostici, come la prova serologica per gli anticorpi anti-gliadina e la gastroscopia con biopsie del duodeno. Inoltre risulta essere più contestualizzata e offre informazioni più precise basate su risposte precedenti dei medici.

- La risposta senza RAG: Suggerisce di consultare un medico specialista senza entrare nei dettagli specifici dei test diagnostici. È più generica e meno informata, poiché non utilizza il contesto fornito dalle risposte precedenti.

Capitolo 3: Implementazione Interfaccia

3.1 Creazione dell'interfaccia per l'utente finale

Per offrire un'interfaccia utente interattiva e facile da usare per il nostro sistema di risposta automatica, abbiamo utilizzato la libreria **gradio**, che permette di creare interfacce web che possono essere utilizzate per interagire con modelli di machine learning in modo semplice e intuitivo. L'interfaccia è composta dai seguenti elementi:

- *Menu a tendina per selezionare il sito*: L'utente può selezionare il sito di interesse da un elenco predefinito.
- *Area di testo per inserire la domanda*: L'utente può digitare la domanda che desidera porre al modello.
- *Pulsante per generare la risposta*: Una volta inserita la domanda e selezionato il sito, l'utente può cliccare un pulsante per generare la risposta.

3.2 Personalizzazione dell'interfaccia

Per migliorare e leggittimare l'esperienza dell'utente e adattare l'interfaccia alle esigenze specifiche del progetto, abbiamo personalizzato vari aspetti.

Integrazione del menù a tendina

Introdurre un menu a tendina per selezionare il sito di interesse può essere vantaggioso per facilitare la navigazione di un utente permettendogli semplicemente di selezionare il sito desiderato dall'elenco proposto. Inoltre offre opzioni chiare e da consapevolezza all'utente della provenienza delle risposte ottenute, migliorandone l'esperienza complessiva.

Integrazione del menù di settaggio data

Il menù del settaggio data serve per poter ottenere delle risposte dal modello, che sono basate sui consulti risalenti alla data specificata. Può essere particolarmente utile per domande su epidemie stagionali, trattamenti innovativi e altre situazioni temporanee. Consente inoltre di analizzare come le risposte e le pratiche mediche siano cambiate nel tempo, fornendo insight su trend emergenti e aggiornamenti nelle linee guida mediche.

Capitolo 4: Analisi dei Dati

Prima analisi: Serie Temporale

La prima analisi consiste nella generazione e visualizzazione di una serie temporale che mostra come il numero di documenti varia nel tempo per ciascun sito.

Utilizzando la funzione create_time_series, i documenti presenti nel database vengono analizzati e organizzati in modo da poter rappresentare graficamente il loro conteggio nel corso del tempo. Successivamente, la funzione plot_time_series utilizza questi dati per costruire un grafico che illustra l'evoluzione temporale del numero di documenti per ogni sito. Infine, la funzione plot restituisce il grafico generato, permettendo all'utente finale di visualizzare facilmente le informazioni. Questo processo consente di ottenere una visione chiara e dettagliata delle dinamiche temporali dei documenti, evidenziando eventuali tendenze o fluttuazioni significative nel tempo.

Seconda analisi: Numero di post per categoria medica

La seconda analisi esamina il numero di post associati a diverse categorie mediche sui siti Dica33 e MedicItalia. Viene effettuato un conteggio dei post per ogni categoria, fornendo una panoramica quantitativa della distribuzione dei post su argomenti specifici.

Questa analisi consente di identificare quali categorie mediche ricevono maggiore approfondimento nel sistema, offrendo preziose informazioni per comprendere le aree maggiormente informate.

Conclusioni

Il progetto ha implementato un sistema avanzato di web scraping e analisi dei dati per due importanti siti medici italiani, Dica33 e MedicItalia. La raccolta e il processamento parallelo dei link tramite un pool di driver Chrome hanno permesso di gestire efficacemente il web scraping, riducendo significativamente i tempi di elaborazione. Questo ha portato alla raccolta di un ampio dataset di domande e risposte mediche, archiviato in file pickle per ulteriori analisi.

Risultati Raggiunti

- Raccolta dei Dati: Sono stati raccolti 11.923 dati da Dica33 e 18.828 da MedicItalia, per un totale di 30.751 dati, garantendo una copertura ampia e approfondita di vari argomenti medici.
- Preelaborazione dei Dati: Le operazioni di pulizia, filtraggio e trasformazione hanno prodotto testi strutturati e pronti per l'analisi, migliorandoqualità e accuratezza dei dati.
- Archiviazione dei Dati: Utilizzando MongoDB, i dati sono stati memorizzati in modo efficiente, con rappresentazioni vettoriali che facilitano le ricerche avanzate. La struttura del database garantisce flessibilità e facilità di aggiornamento, supportando future estensioni del progetto.
- Integrazione del Modello LLM: Il modello *google/gemma-1.1-7b-it* è stato implementato per generare risposte basate su un contesto preciso e istruzioni dettagliate. La combinazione di ricerca vettoriale e generazione di risposte (RAG) ha migliorato la pertinenza e la qualità delle risposte fornite agli utenti.
- Interfaccia Utente: basata su Gradio rende il sistema accessibile e facile da usare.

Sviluppi Futuri

Sono stati individuati dei migliormenti apportabili al sistema di Q&A per aumentarne l'efficienza e l'usabilità.

Integrazione degli URL di riferimento del contesto

Abbiamo pensato di sfruttare ulteriormente l'infomrazione ottenuta con il Web Scraping per ogni post , relativa al sito di provenienza. Potrebbe essere sfruttata permettendoci di allegare al consulto una fonte di riferimento tramite la quale l'utente possa avere una prospettiva più ampia del contesto e provenienza della risposta.

Filtraggio per Categorie

La possibilità di indicare specifiche categorie da considerare consentirà un filtraggio più preciso delle informazioni, assicurando che le risposte siano maggiormente pertinenti e mirate alle necessità dell'utente. Le categorie saranno predefinite e selezionabili tramite un menu a tendina, offrendo un'esperienza utente intuitiva e personalizzata.

Estensione del Dataset

Continuare a raccogliere e aggiornare i dati per mantenere il sistema aggiornato con le ultime informazioni mediche e ampliare le categorie e le dimensioni.