

PROGETTO TEXT MINING

CREAZIONE DI UN CHATBOT IN GRADO DI PROCESSARE DATASET IN ITALIANO

MEMBRI DEL GRUPPO:

ARENA LETIZIA M63001513

FERRARA LEONARDO M63001517

RUSSO DAVIDE M63001607

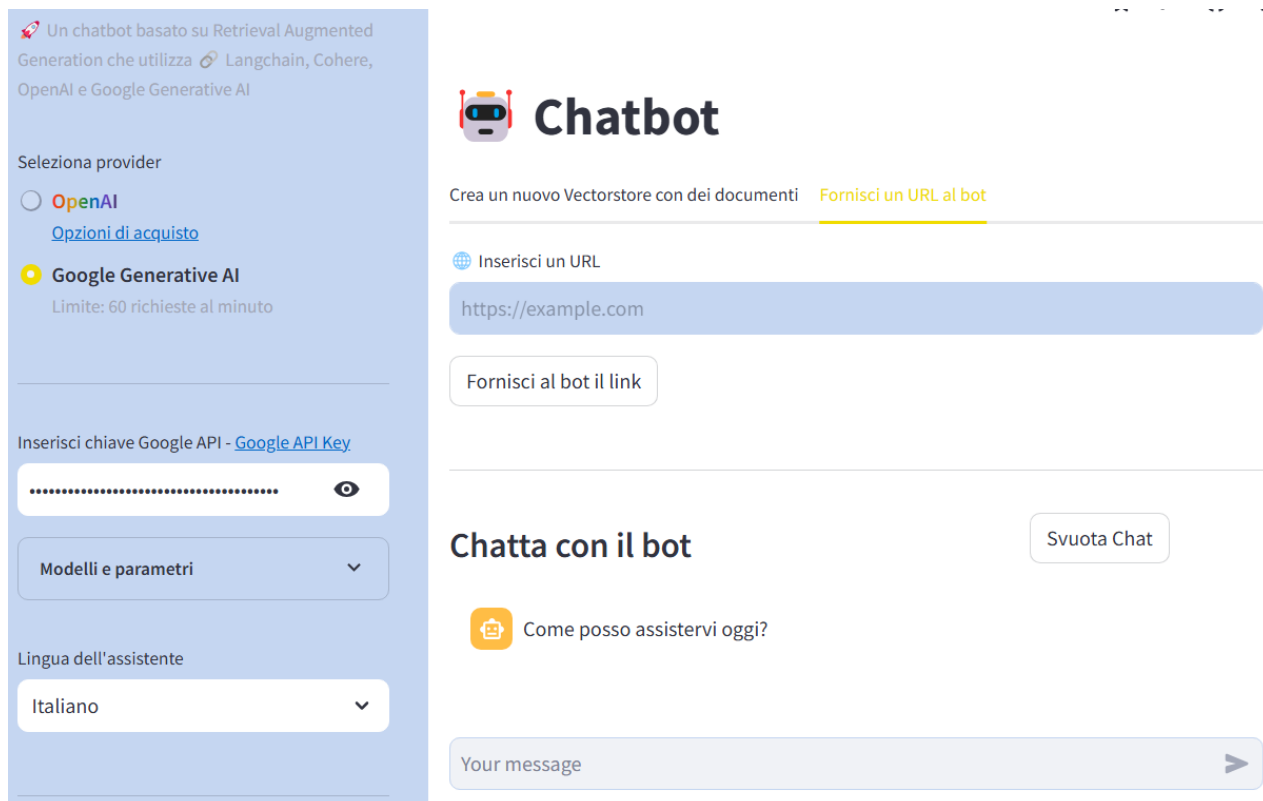
INTRODUZIONE

Il chatbot è disponibile al link <https://textminingchat.streamlit.app/>.

Per la creazione del chatbot si è scelto di utilizzare il framework Streamlit ed implementare un chatbot basato su Retrieval-Augmented Generation (RAG) utilizzando Langchain.

Il codice è hostato su GitHub (repository <https://github.com/learena/Bot>) e permette l'utilizzo sia di OpenAI che di Google GenerativeAI come modelli di linguaggio. Il bot è in grado di processare dataset e rispondere in 10 differenti lingue, e permette di scegliere tra 3 retriever differenti.

È possibile caricare documenti di tipo PDF, TXT, e CSV, ed inoltre è presente la funzione di caricare pagine web mediante URL. La gestione dei documenti avviene mediante ChromaDB.



The screenshot displays the 'Text Mining Chatbot' interface, which is a web application built with Streamlit. The interface is divided into two main sections: a configuration sidebar on the left and a main chat area on the right.

Configuration Sidebar (Left):

- Header:** A rocket icon followed by the text: "Un chatbot basato su Retrieval Augmented Generation che utilizza Langchain, Cohere, OpenAI e Google Generative AI".
- Selezione provider:** Two radio buttons are present. The first is labeled "OpenAI" with a link "Opzioni di acquisto" below it. The second is selected and labeled "Google Generative AI" with the text "Limite: 60 richieste al minuto" below it.
- API Key:** A text input field labeled "Inserisci chiave Google API - Google API Key" with a masked key "....." and an eye icon to toggle visibility.
- Modelli e parametri:** A dropdown menu currently showing "Modelli e parametri".
- Lingua dell'assistente:** A dropdown menu currently showing "Italiano".

Main Chat Area (Right):

- Header:** A robot icon followed by the title "Chatbot".
- Actions:** Two buttons are available: "Crea un nuovo Vectorstore con dei documenti" and "Fornisci un URL al bot" (highlighted in yellow).
- URL Input:** A text input field labeled "Inserisci un URL" containing the example "https://example.com". Below it is a button labeled "Fornisci al bot il link".
- Chat Section:** A section titled "Chatta con il bot" with a "Svuota Chat" button.
- Example Prompt:** A button with a speech bubble icon and the text "Come posso assistervi oggi?".
- Message Input:** A large text input field at the bottom labeled "Your message" with a send button (arrow icon) on the right.

DESCRIZIONE DEL CODICE (BOT/RAG_APP.PY)

IMPORT

La prima parte del file importa le librerie necessarie, tra cui di principale importanza abbiamo:

- **Streamlit** per la creazione dell'interfaccia grafica.
- **OpenAI** per interagire con l'API del modello GPT.
- **GoogleAI** per interagire con l'API del modello Google.
- **Chroma** per la gestione dell'indice di documenti.
- **PyPDF2** per estrarre testo dai PDF caricati.
- **OS** per la gestione delle variabili d'ambiente.
- **Langchain** per l'integrazione dei modelli linguistici nell'applicazione.

CONFIGURAZIONE CHIAVI API E ALTRE OPZIONI

Il codice utilizza tre chiavi API:

- OPENAI_KEY: per interagire con i modelli di OpenAI.
- GOOGLE_KEY: per interagire con i modelli di Google Generative AI.
- COHERE_KEY: per interagire con il retriever Cohere Reranker.

Le chiavi devono essere configurate tramite variabili d'ambiente per la corretta esecuzione dell'app: in particolare, GOOGLE_KEY e COHERE_KEY sono salvate nel file .env della repository Bot su Github, mentre OPENAI_KEY, per motivi di sicurezza, è salvata all'interno della sezione "Segreti" di Streamlit.

A seguito della configurazione delle chiavi si ha una breve sezione dedicata alla configurazione dei servizi LLM (list_LLM_providers), della lingua dell'assistente (dict_welcome_message) e dei tipi di retriever (list_retriever_types).

CREAZIONE DELL'INTERFACCIA STREAMLIT

Si configura in primo luogo la pagina Streamlit utilizzando "Conversa con il bot" come nome per l'applicazione, e si imposta il titolo visibile all'utente. Si inizializzano le chiavi API necessarie per OpenAI, Google e Cohere come variabili nello stato della sessione di Streamlit.

Si definisce poi una funzione `expander_model_parameters()` (sezione "Modelli e parametri" nell'app), che

1. Aggiunge un campo di input per la chiave API, specifico per il provider selezionato. Questo campo è completato per default dai valori di chiave impostati precedentemente.

2. Implementa un expander per configurare i parametri del modello:
 - Selezione del modello: "gpt-3.5-turbo-0125", "gpt-3.5-turbo", "gpt-4-turbo-preview" per OpenAI e "gemini-pro" per Google.
 - Temperatura: Controlla la casualità delle risposte. Valori più alti aumentano la creatività delle risposte.
 - Top-p: Regola la probabilità cumulativa per il campionamento e controlla quanto è deterministico il modello. Valori più alti aumentano la creatività delle risposte.

In questa sezione viene inoltre definita la funzione `sidebar_and_documentChooser()`, che configura l'interfaccia della barra laterale e le schede principali dell'app.

All'interno della barra laterale si ha:

- Una breve descrizione del chatbot
- Un chooser che consente di scegliere il provider LLM (OpenAI o Google)
- Un selettore per la lingua dell'assistente
- Un chooser per scegliere il tipo di retriever da utilizzare (Vectorstore backed retriever, Cohere reranker o Contextual compression)

Nella scheda principale sono disponibili due tab:

1. Tab: "Crea un nuovo Vectorstore con dei documenti"

- Permette all'utente di caricare file (PDF, TXT, DOCX, CSV).
- Consente di specificare un nome per il vectorstore (Chroma DB).
- Fornisce un pulsante per creare un vectorstore processando i documenti caricati.

2. Tab: "Fornisci un URL al bot"

- Aggiunge un campo di input per un URL da cui il bot può recuperare informazioni.
- Include un pulsante per inviare l'URL e processarlo.

PROCESSING DEI DOCUMENTI E CREAZIONE DEL VECTORSTORE (CHROMA DB)

In questa sezione sono presenti le seguenti funzioni:

- `delte_temp_files()`: questa funzione elimina i file temporanei dalla directory `./data/tmp`. Viene utilizzata per assicurarsi che non ci siano file residui che potrebbero interferire con il caricamento di nuovi documenti.
- `langchain_document_loader()`: crea una lista di documenti da salvare nella directory temporanea `TMP_DIR`. Utilizza loader specifici a seconda del tipo di documento da salvare.
- `split_documents_to_chunks()`: divide i documenti in chunk utilizzando `RecursiveCharacterTextSplitter`.

- `select_embeddings_model()`: seleziona il modello di embedding (OpenAI o Google) dove per embedding si intende delle rappresentazioni numeriche dei documenti. Le chiavi API necessarie vengono recuperate dalla sessione (`st.session_state`).
- `create_retriever()`: crea il retriever, ossia l'interfaccia che ritorna documenti a partire da una query non strutturata. Il retriever può essere configurato in tre modalità principali:
 - Vectorstore backed retriever: è il retriever di base, recupera documenti basati sulla similarità vettoriale. È configurato tramite `Vectorstore_backed_retriever()`.
 - Contextual compression retriever: è un retriever che fa il wrapping del retriever di base in un `ContextualCompressionRetriever`. Questo compressore è un Compressor Pipeline, che divide i documenti in chunk più piccoli, rimuove documenti ridondanti, filtra i documenti più rilevanti, e riordina i documenti in maniera tale che i più rilevanti siano all'inizio/alla fine della lista. È configurato tramite `create_compression_retriever()`.
 - Cohere reranker: utilizza un modello di re-ranking di Cohere per riordinare i documenti in base alla rilevanza. È configurato tramite `CohereRerank_retriever()`.
- `Vectorstore_backed_retriever()`: crea un retriever vectorstore-backed.
- `Create_compression_retriever()`: costruisce un retriever avanzato con un approccio di compressione contestuale utilizzando gli step visti prima. Combina gli step in una pipeline `DocumentCompressorPipeline`.
- `CohereRerank_retriever()`: costruisce un `ContextualCompressionRetriever` utilizzando il CohereRerank endpoint (servizio di re-ranking di Cohere) per riordinare i documenti basati sulla rilevanza rispetto alla domanda.
- `submit_url()`: carica i documenti da un URL specificato dall'utente. In primo luogo, controlla che l'input sia valido, poi elimina i file temporanei precedenti ed utilizza `WebBaseLoader` per caricare il contenuto della pagina web. Divide i documenti in chunk, seleziona il modello di embedding e memorizza i dati in un vectorstore (Chroma dB). Configura poi un retriever basato sui parametri dell'utente e crea una memoria ed una catena di retrieval (`ConversationalRetrievalChain`) per rispondere a domande sull'URL.
- `chain_RAG_blocks()`: compie gli stessi passaggi della funzione precedente ma nel caso in cui il documento sia stato caricato direttamente da locale. Si nota che il sistema RAG è composto da Retrieval (che include i document loader, il text splitter, il vectorstore ed il retriever), Memoria e Conversational Retrieval Chain (contiene la lista di tutti i messaggi precedenti e la nuova domanda).

Nel caso in cui si stia eseguendo il codice in locale, il vectorstore viene salvato localmente in una directory specifica (`LOCAL_VECTOR_STORE_DIR`), utilizzando il nome scelto dall'utente; nel caso dell'applicazione deployata non viene invece garantita persistenza.

Si nota che in queste funzioni viene spesso fatta la verifica degli input, per cui sono implementati vari controlli per assicurarsi che l'utente abbia fornito tutte le chiavi API

richieste e che i file/documenti siano corretti.

Si nota inoltre che i dati relativi a retriever, vectorstore e memoria sono salvati in `st.session_state`, una funzionalità di Streamlit per gestire lo stato tra le interazioni dell'utente.

CREAZIONE DELLA MEMORIA

La memoria viene creata tramite la funzione `create_memory()`, che crea un oggetto di memoria per gestire le conversazioni, in grado di conservare il contesto delle conversazioni, permettendo al sistema di rispondere in modo più coerente. Il tipo di memoria creato dipende dal modello di linguaggio specificato (es. gpt-3.5-turbo o un altro modello). La memoria viene creata con `ConversationSummaryBufferMemory()` nel caso in cui si stia utilizzando gpt-3.5-turbo (è un tipo speciale di memoria progettata per riassumere automaticamente la cronologia della conversazione, in modo da rimanere entro i limiti di token del modello), e con `ConversationBufferMemory()` per tutti gli altri modelli (è una versione più semplice della memoria che registra tutta la cronologia della conversazione senza riassunti o limiti di token). Alla fine, la funzione restituisce l'oggetto memoria creato.

CREAZIONE DELLA CONVERSATIONAL RETRIEVAL CHAIN

In questa sezione sono presenti le seguenti funzioni:

- `answer_template()`: crea un template strutturato che istruisce il modello a rispondere usando solo i dati forniti nel contesto, fornendo la cronologia della chat (`chat_history`), il contesto recuperato da un retriever (`context`), la domanda dell'utente (`question`), e la lingua in cui il modello deve rispondere.
- `create_ConversationalRetrievalChain()`: crea una `ConversationalRetrievalChain` seguendo i seguenti passaggi:
 - Condensazione delle domande: invio della domanda di follow-up insieme alla cronologia della chat ad una `condense_question_llm` che parafrasa la domanda e genera una query singola.
 - Definizione dell'answer prompt: combina la domanda autonoma, la cronologia della chat, ed il contesto recuperato dal retriever.
 - Creazione della memoria tramite `create_memory()`.
 - Istanziamento LLM: definisce il modello `standalone_query_generation_llm` per condensare le domande di follow-up ed il modello `response_generation_llm` per generare le risposte.
 - Creazione della `ConversationalRetrievalChain`.

La funzione restituisce infine la catena conversazionale (`chain`) e l'oggetto memoria (`memory`).

- `clear_chat_history()`: reinizializza i messaggi di chat con un messaggio di benvenuto nella lingua selezionata e cancella la memoria, se esiste.
- `get_response_from_LLM()`: gestisce il processo di interazione con la `ConversationalRetrievalChain` per ottenere una risposta dal modello e visualizzarla seguendo i seguenti passaggi:
 - Invocazione del LLM: passa il prompt della domanda (question) alla catena, ottenendo una risposta e i documenti di riferimento.
 - Display dei risultati: aggiorna la cronologia della chat (salvando la domanda dell'utente e la risposta del modello) e mostra la risposta generata ed i documenti di riferimento.

CHATBOT

La funzione finale è la funzione `chatbot()`, che gestisce la logica del bot.

In primo luogo si chiamano le funzioni definite in precedenza che compongono l'interfaccia del chatbot; si passa poi all'inizializzazione della cronologia dei messaggi, per cui se la chiave "messages" non è ancora stata definita in `st.session_state`, viene inizializzata con un messaggio di benvenuto (`dict_welcome_message`) nella lingua selezionata dall'utente. Si crea poi una di input in cui l'utente può inserire il proprio messaggio: se l'utente inserisce del testo, il valore del testo viene salvato nella variabile `prompt`. Questo prompt viene inviato al modello chiamando la funzione `get_response_from_LLM` per ottenere una risposta.

Si arriva infine al blocco principale (`__main__`): che, quando il file viene eseguito, avvia direttamente la funzione `chatbot()`.

Come considerazione finale, si può notare che l'app è progettata per essere modulare e scalabile, con possibilità di estensioni future (ad esempio, aggiungendo il supporto per ulteriori formati di file o modelli).

DEPLOYMENT SU STREAMLIT

La struttura complessiva della repository risulta quindi quella mostrata di fianco. Per il deployment su Streamlit è bastato creare una nuova app inserendo il collegamento alla repository Github indicando il main file path e l'URL scelta.

Si è scelto inoltre di utilizzare un tema personalizzato di Streamlit, in maniera da rendere la pagina più piacevole. La nuova configurazione può essere vista nella cartella `.streamlit`.

