tInstituto Tecnológico y de Estudios Superiores de Monterrey
Campus Guadalajara

Desarrollo e implantación de sistemas de software (Gpo 102)

Actividad 7 - RAG for LLMs and Chatbots

Paola Félix Torres	A00227869
Javier Eric Hernández Garza	A01635390
Arturo Ramos Viedas	A01636133
Karen Priscila Navarro Arroyo	A01641532
Israel Vidal Paredes	A01750543

25/05/2024

Hicimos un código ejemplo de como podríamos hacer un rag usando langachain, OpenAI, Python y Chroma.

```
bs4_strainer = bs4.SoupStrainer(class_=("mw-page-title-main", "page__main"))


loader = WebBaseLoader(
    web_paths=("https://tardis.fandom.com/wiki/Fifteenth_Doctor",
               "https://tardis.fandom.com/wiki/Fourteenth_Doctor",
               "https://tardis.fandom.com/wiki/Doctor",
               "https://tardis.fandom.com/wiki/Thirteenth_Doctor"),
    bs_kwargs={"parse_only": bs4_strainer},
)
docs = loader.load()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200, add_start_index=True)
all_splits = text_splitter.split_documents(docs)

vectorstore = Chroma.from_documents(documents=all_splits, embedding=OpenAIEmbeddings())
```

Extracción de Contenido de Páginas Web
Usamos bs4.SoupStrainer para filtrar solo ciertos elementos HTML de las páginas web especificadas. WebBaseLoader carga el contenido de las páginas web especificadas utilizando los filtros definidos por bs4_strainer y luego el contenido cargado se divide en partes más pequeñas usando RecursiveCharacterTextSplitter, lo cual es útil para manejar grandes volúmenes de texto.

```
vectorstore = Chroma.from_documents(documents=all_splits, embedding=OpenAIEmbeddings())

retriever = vectorstore.as_retriever(search_type="similarity", search_kwargs={"k": 6})
query = input("Ask a doctor who question: ")
retrieved_docs = retriever.invoke(query)

api_key = os.getenv('OPENAI_API_KEY')
llm = ChatOpenAI(model="gpt-3.5-turbo-0125", api_key=api_key)
prompt = hub.pull("rlm/rag-prompt")
example_messages = prompt.invoke({"context": "filler context", "question": "filler question"}).to_message
print(example_messages)

def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()      You, 20 hours ago • a
)
```

Despues hicimos el procesamiento del lenguaje creando un Chroma vector store a partir de los documentos divididos, utilizando embeddings de OpenAI. El siguiente paso fue usar retriever que se configuro para buscar similitudes entre consultas y documentos almacenados.

Los utlimos pasos son:

- Invocamos el retriever con la consulta proporcionada para encontrar documentos relevantes.
- Preparamos un mensaje del usuario que utiliza un prompt específico obtenido de hub.pull.

- Luego definimos uno cadena de procesamiento que incluye el formateo de los documentos, la generación de mensajes basados en el contexto y la pregunta, y la ejecución de estas acciones mediante el modelo de chat.

Cadena de Procesamiento Final

- La rag_chain combina todas las etapas que describimos anteriormente, para hacer una integraciónde diversas herramientas y técnicas para extraer información relevante de fuentes en línea, procesarla y generar respuestas a preguntas específicas utilizando modelos de lenguaje avanzados.