2b_Openai_RAG

November 12, 2024

OPENAI RAG LLM setup

Loading packages, libraries and secrets into notebook

[ ]:
```
# Importing the required libraries
from pymongo import MongoClient
from langchain_mongodb import MongoDBAtlasVectorSearch
from langchain_openai import OpenAI
from langchain_openai import ChatOpenAI
from langchain.chains import RetrievalQA
import gradio as gr
from gradio.themes.base import Base
from langchain_core.output_parsers import StrOutputParser
from langchain.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnableParallel, RunnablePassthrough
from sentence_transformers import SentenceTransformer # https://huggingface.co/
 ↪thenlper/gte-large
import os
from dotenv import load_dotenv
```

[ ]:
```
# Accessing the secrets from the environment variables
load_dotenv()
MONGO_URI_SQL = os.getenv("MONGO_URI_SQL")
MONGO_URI_schema = os.getenv("MONGO_URI_Schema")
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
HF_Token = os.getenv("HF_TOKEN")
```

Generating the embedding

[ ]:
```
# Embedding model setup
embedding_model = SentenceTransformer("thenlper/gte-large")

class CustomEmbeddingFunction:
    def __init__(self, model):
        self.model = model

    def embed_documents(self, texts):
        """Embeds a list of documents."""
        embeddings = self.model.encode(texts)
```

```
        return embeddings.tolist()

    def embed_query(self, text):
        """Embeds a single query."""
        embedding = self.model.encode(text)
        return embedding.tolist()

# Wrap the SentenceTransformer model
embedding_function = CustomEmbeddingFunction(embedding_model)
```

[ ]:
```
## MongoDB setup
# SQL Vector
client_SQL = MongoClient(MONGO_URI_SQL)
dbName_SQL = "MVector"
collectionName_SQL = "MTSQL"
collection_SQL = client_SQL[dbName_SQL][collectionName_SQL]
index_name_SQL = "vector_index_SQL"

## SQL Vector setup
# Vector store setup
vector_store_SQL = MongoDBAtlasVectorSearch(
    client=client_SQL,
    database=dbName_SQL,
    collection=collection_SQL,
    index_name=index_name_SQL,
    embedding=embedding_function,
    text_key="Query"
)

# Retriever setup
retriever_SQL = vector_store_SQL.as_retriever(search_kwargs={"k": 4})

# Define a custom logging retriever to see what the retriever is passing on
class LoggingRetrieverSQL:
    def __init__(self, retriever_SQL):
        self.retriever_SQL = retriever_SQL

    def __call__(self, query):
        # Retrieve the documents
        documents_SQL = self.retriever_SQL.invoke(query)

        # Log or print the retrieved documents
        print("Retrieved Documents:")
        for doc in documents_SQL:
            print(doc)

        # Return the retrieved documents
```

```
        return documents_SQL

# Wrap your retriever with the logging retriever
logging_retriever_SQL = LoggingRetrieverSQL(retriever_SQL)
```

Chain setup

[ ]:
```
query = "SELECT T1.Name FROM conductor AS T1 JOIN orchestra AS T2 ON T1.
 ↪Conductor_ID  =  T2.Conductor_ID GROUP BY T2.Conductor_ID ORDER BY COUNT(*)␣
 ↪DESC LIMIT 1"
output_length = len(query.split())*3 # word count of SQL query multiplied by␣
 ↪four

# Model and parsing setup
model = ChatOpenAI(api_key=OPENAI_API_KEY, model="gpt-4o-mini", temperature = 0)
parser = StrOutputParser()


# Define prompt template
template = """
Provide first a natural language Translation followed by an Explanation of the␣
 ↪SQL Query. Go through it step by step and output the result in simple and␣
 ↪concise language. Use the information of the Context as examples for the␣
 ↪translation. Keep the output in line with the Length number.

Context: {context}

Query: {query}

Lenght: {output_length}
"""

prompt = ChatPromptTemplate.from_template(template)

# Chain setup
chain_2b = (
    {"context": logging_retriever_SQL, "query": RunnablePassthrough(),␣
 ↪"output_length" : RunnablePassthrough()}
    | prompt
    | model
    | parser
)

# Execute the chain with the logging retriever
chain_2b.invoke(query)
```

Chat interface setup

Markdown format of Chat interface setup for testing.

Change cell type below to Python, when running only this script.

[ ]:
```python
# Define the chain_invoke function
def chain_2b_invoke(query):
    # Execute the chain with the logging retriever
    result = chain_2b.invoke(query)
    # Return the result
    return result

# Create a web interface for the app, using Gradio
with gr.Blocks(theme=Base(), title="Question Answering App using Vector Search␣
 ↪+ RAG") as demo:
    gr.Markdown(
        """
        # Question Answering App using Atlas Vector Search + RAG Architecture
        """)
    textbox = gr.Textbox(label="Enter your SQL statement:")
    with gr.Row():
        button = gr.Button("Submit", variant="primary")
    output = gr.Textbox(lines=1, max_lines=30, label="Natural language␣
 ↪translation and explanation:")

# Call chain_invoke function upon clicking the Submit button
    button.click(chain_2b_invoke, textbox, outputs=output)

demo.launch()
```