sbiannualinsightsbot

February 3, 2024

1 SBI FinInsight Bot

This project, titled “SBIFinInsight Bot,” is submitted by **Anupam Dhiman (P2023PTLP0040)** in fulfillment of the graded assignment for the course Python for Data Science at Plaksha University. The objective of this project is to develop a Vector Indexed Retrieval Augmented Generation (RAG) based Question-Answering (QA) Chatbot using Pretrained Transformer models. The focus is on the finance domain, specifically leveraging insights from the State Bank of India (SBI) annual report for the fiscal year 2022-23.

BUSINESS PROBLEM AND RELEVANCE

The SBI FinInsight Bot project addresses a critical challenge inherent in navigating extensive financial documentation, specifically the State Bank of India’s (SBI) annual report for the fiscal year 2022-23. Annual reports, known for their comprehensive nature, present a formidable barrier to efficient information retrieval. Manually sifting through voluminous documents for specific details is not only time-consuming but also poses challenges in ensuring accuracy and compliance. To tackle thistheur project introduces a groundbreaking solution: a Vector Indexed Retrieval Augmented Generation (RAG) based QA Chatbot powered by Pretrained Transformer models tailored to the finance domain.

Key Business Problems Addressed:

1. **Efficient Information Retrieval:**
    - The SBI annual report is a repository of diverse financial information, from performance metrics to strategic insights. Extracting pertinent details manually is arduous and time-intensive. The SBI FinInsight Bot streamlines this process, offering a responsive interface that efficiently retrieves specific information from the annual report.
2. **Enhanced User Experience:**
    - Stakeholders, investors, and analysts often face challenges in accessing relevant financial data within annual reports. The FinInsight Bot transforms this experience by providing a user-friendly interface that allows stakeholders to interact naturally. Users can pose queries in plain language, making information retrieval intuitive and efficient.
3. **Improved Decision-Making:**
    - Quick access to relevant financial insights is paramount for informed decision-making. The bot facilitates this by promptly providing accurate information. Decision-makers can evaluate the bank’s financial health, performance against targets, and strategic initiatives with ease, fostering better decision-making processes.

4. **Scalability and Accessies scalability. Operating 24/7, it transcends time zones, making information accessible to a global audience. This not only enhances user accessibility but also caters to stakeholders' diverse tim5 constraints.

5. **Compliance and Accuracy:**

   - Leveraging pretrained transformer models ensures that the FinInsight Bot's responses are grounded in accurate information extracted directly from the annual report. This adherence to accuracy is vital for compliance with regulatory requirements governing the dissemination of finan6ial information.

6. **Competitive Advantage:**

   - The implementation of an advanced chatbot utilizing state-of-the-art natural language processing techniques positions the organization at a competitive advantage. This strategic move showcases a commitment to technological innovation, enhancing stakeholder communication, and fostering transparency in the dissemination of he dynamic finance domain.

2 DATA COLLECTION METHODOLOGY

the FinInsight Bot is trained on the SBI Annual Report 2022-23 in Pdf format.The detailed report is in the folder titled "Data" that is submitted along with this report.

The following files have been submitted along with this report:

ingest.py

model.py

The code snippets of these files will be explained subsequently in detail.

2.1 ingest.py

This script is designed to load PDF documents from a specified directory, split the text into smaller chunks, use Hugging Face embeddings, and create a FAISS vector database from the processed texts. The resulting vector database is then saved locally.

2.1.1 Imports

[1]:
```python
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
from langchain.document_loaders import PyPDFLoader, DirectoryLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

These lines import various modules from the langchain package. The code is using functionality related to embeddings, vector stores, document loading, and text splitting from the langchain library.

2.1.2 Global variables

[2]:
```
DATA_PATH = "C:/Users/anupa/OneDrive/PROJECTS/python_project/data/"
DB_FAISS_PATH = "C:/Users/anupa/OneDrive/PROJECTS/python_project/
 ↪vectorstoredb_faiss"
```

These are global variables specifying the paths to the data and the location where the FAISS vector database will be stored.

2.1.3 Function: create_vector_db()

[3]:
```
def create_vector_db():
    loader = DirectoryLoader(DATA_PATH,
                             glob='*.pdf',
                             loader_cls=PyPDFLoader)
```

The create_vector_db function initializes a DirectoryLoader object from the langchain library, which loads documents from a specified directory (DATA_PATH) with a given file extension (*.pdf) using the PyPDFLoader class.

2.1.4 Retrieve loaded document

[ ]:
```
documents = loader.load()
```

The load() method is then called on the loader object to retrieve a list of loaded documents.

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
                                               chunk_overlap=50)
texts = text_splitter.split_documents(document)
```

A RecursiveCharacterTextSplitter is initialized with specific chunk size and overlap parameters. The split_documents method is then called to break down the loaded documents into smaller text chunks.

[ ]:
```
embeddings = HuggingFaceEmbeddings(model_name='sentence-transformers/
 ↪all-MiniLM-L6-v2',
                                   model_kwargs={'device': 'cpu'})
```

An instance of HuggingFaceEmbeddings is created with a specific model name and device (‘cpu’ in this case).

[ ]:
```
db = FAISS.from_documents(texts, embeddings)
    db.save_local(DB_FAISS_PATH)
```

A FAISS vector database (db) is created from the processed texts and embeddings using the FAISS.from_documents method. The resulting database is then saved locally to the specified path (DB_FAISS_PATH).

2.1.5 Main Execution:

```
[ ]: if __name__ == "__main__":
         create_vector_db()
```

The create_vector_db function is called only if the script is executed as the main module (not imported as a module in another script). This ensures that the database creation process is triggered when the script is run directly.

2.2 model.py

The model.py script orchestrates the initialization, deployment, and interaction flow of the SBIAnnualInsight Bot using pretrained transformer models, embeddings, and a vector database. It integrates with ChainLit to create a seamless chatbot experience, offering an asynchronous and user-friendly interface for stakeholders to query financial insights from the SBI annual report.

2.2.1 Imports

```
[ ]: from langchain.document_loaders import PyPDFLoader, DirectoryLoader
     from langchain import PromptTemplate
     from langchain.embeddings import HuggingFaceEmbeddings
     from langchain.vectorstores import FAISS
     from langchain.llms import CTransformers
     from langchain.chains import RetrievalQA
     import chainlit as cl
```

These lines import various modules and classes from the langchain and chainlit packages, which are used for document loading, embeddings, vector stores, language models (llms), and the retrieval question-answering chain.

2.2.2 Global Variable

```
[ ]: DB_FAISS_PATH = "C:/Users/anupa/OneDrive/PROJECTS/python_project/
       ↪vectorstoredb_faiss"
```

This global variable specifies the path to the FAISS vector database

2.2.3 Custom Prompt Template:

```
[ ]: custom_prompt_template = """Use the following pieces of information to answer␣
       ↪the user's question.
     If you don't know the answer, just say that you don't know, don't try to make␣
       ↪up an answer.

     Context: {context}
     Question: {question}

     Only return the helpful answer below and nothing else.
```

```
Helpful answer:
"""
```

This variable defines a custom prompt template to be used for Question-Answering (QA) retrieval.

2.2.4 Functions):

[ ]:
```
def set_custom_prompt():
    prompt = PromptTemplate(template=custom_prompt_template,
                            input_variables=['context', 'question'])
    return prompt
```

This function creates a PromptTemplate object using the custom prompt template.

[ ]:
```
def retrieval_qa_chain(llm, prompt, db):
    qa_chain = RetrievalQA.from_chain_type(llm=llm,
                                           chain_type='stuff',
                                           retriever=db.
 ↪as_retriever(search_kwargs={'k': 2}),
                                           return_source_documents=True,
                                           chain_type_kwargs={'prompt': prompt}
                                           )
    return qa_chain
```

This function initializes a RetrievalQA chain using a specified language model (llm), a custom prompt, and a FAISS vector database (db). It configures the chain to return source documents along with the answer.

[ ]:
```
def load_llm():
    llm = CTransformers(
        model='abhishek/llama-2-7b-hf-small-shards',
        model_type="llama",
        max_new_tokens=512,
        temperature=0.5
    )
    return llm
```

This function loads a language model (llm) using the CTransformers class from the langchain package.

[ ]:
```
def qa_bot():
    embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/
 ↪all-MiniLM-L6-v2",
                                           model_kwargs={'device': 'cpu'})
    db = FAISS.load_local(DB_FAISS_PATH, embeddings)
    llm = load_llm()
    qa_prompt = set_custom_prompt()
    qa = retrieval_qa_chain(llm, qa_prompt, db)
    return qa
```

This function initializes the QA bot by creating embeddings, loading the FAISS vector database, loading the language model, setting the custom prompt, and creating the QA chain.

[ ]:
```
def final_result(query):
    qa_result = qa_bot()
    response = qa_result({'query': query})
    return response
```

This function takes a query as input, invokes the QA bot, and returns the response.

2.2.5 ChainLit Code

[ ]:
```
@cl.on_chat_start
async def start():
    chain = qa_bot()
    msg = cl.Message(content="Starting the bot...")
    await msg.send()
    msg.content = "Hi, Welcome to Finance Bot. What is your query?"
    await msg.update()

    cl.user_session.set("chain", chain)
```

This asynchronous function is executed when the chat starts. It initializes the QA bot and sets it in the user's session.

[ ]:
```
@cl.on_message
async def main(message):
    chain = cl.user_session.get("chain")
    cb = cl.AsyncLangchainCallbackHandler(
        stream_final_answer=True, answer_prefix_tokens=["FINAL", "ANSWER"]
    )
    cb.answer_reached = True
    res = await chain.acall(message, callbacks=[cb])
    answer = res["result"]
    sources = res["source_documents"]

    if sources:
        answer += f"\nSources:" + str(sources)
    else:
        answer += "\nNo sources found"

    await cl.Message(content=answer).send()
```

This asynchronous function is executed on each received message. It retrieves the QA bot from the user's session, calls the QA chain on the incoming message, and sends back the answer along with any source documents found. If no sources are found, it indicates that in the response.

2.2.6 How to access the SBI FinInsight Bot

1. Store the data in the data folder in .pdf format.
2. Run ingest.py to create the vector database.
3. Run model.py to create the chatbot interface and ask the necessary questions.