AI-poweredHR

June 28, 2025

1 Crafting an AI-Powered HR Assistant: A Use Case for Nestle's HR Policy Documents

2 *Description:*

This is a conversational chatbot that responds to user inquiries using PDF document information.

3 Steps to Perform:

1. Set up the Environment
2. Define a Document Loader
3. Create a Document Splitter
4. Embed the Text and Save it in Vector Stores
5. Create a Retrieval Function
6. Run the Chatbot using Gradio UI and answer queries

4 Step 1: Set up the Environment

- Import the necessary libraries.

[1]:
```
#Import necessary libraries
import os
import openai
import sys
```

[17]:
```
os.environ["OPENAI_API_KEY"] = "sk-abcdef1234567890abcdef1234567890abcdef12"
```

5 Step 2: Define a Document Loader

- Use a document loader like PyPDF to load information from a PDF file.

[1]:
```
#Using PyPDF
from langchain.document_loaders import PyPDFLoader

Doc_loader = PyPDFLoader("HR_policy.pdf")
extracted_text = Doc_loader.load()
```

6 Step 3: Create a Document Splitter

- Break down big pieces of text into smaller parts using text splitters.

[2]:
```
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter  = RecursiveCharacterTextSplitter(
    chunk_size=150,
    chunk_overlap=0,
    separators=["\n\n", "\n", "(?<=\. )", " ", ""]
)
splitted_text=text_splitter.split_documents(extracted_text)
```

7 Step 4: Embed the Text and Save it in Vector Stores

- Arrange a place to store and organize the text splits to make them searchable.
- Employ OpenAIEmbeddings to create a pretrained model instance, saving the results in a specified directory path.
- FAISS for similarity search in the respective chunks

[3]:
```
from langchain.embeddings import OpenAIEmbeddings
embeddings = OpenAIEmbeddings()
```

[4]:
```
from langchain.vectorstores import FAISS
```

[5]:
```
#persist_directory = "chroma_vector"
```

[6]:
```
vectordb = FAISS.from_documents(
    documents=splitted_text,
    embedding=embeddings,
)
```

8 Step 5: Create a Retrieval Function

- Retrieve pertinent data from storage based on user input using a retriever.

[7]:
```
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
```

[25]:
```
from langchain.chains import RetrievalQA
Retriever_chain = RetrievalQA.from_chain_type(llm,
                                      retriever=vectordb.as_retriever(),
                                      return_source_documents=False,
                                      )
```

9 Step 6: Designing a user-friendly Chatbot interface with Gradio

- Set up the chatbot, run it and interact with it.

[18]:
```
#!pip install --upgrade gradio
```

[19]:
```
import gradio as gr
```

[26]:
```
# Define the Gradio chatbot function
def chatbot(message, history):
    if not message.strip():
        return "Please enter a valid question."
    try:
        response = Retriever_chain.run(message)
        return response
    except Exception as e:
        import traceback
        print("Error:", e)
        traceback.print_exc()
        return f"  An error occurred:\n{str(e)}"


# Create the Gradio Interface
chat_ui = gr.ChatInterface(
    fn=chatbot,
    title="HR Policy Assistant",
    description="Ask me anything about the HR Policy.",
    theme="default",
)

# Launch the chatbot
chat_ui.launch(share=True)
```

```
/voc/work/.local/lib/python3.10/site-packages/gradio/chat_interface.py:339:
UserWarning: The 'tuples' format for chatbot messages is deprecated and will be
removed in a future version of Gradio. Please set type='messages' instead, which
uses openai-style 'role' and 'content' keys.
  self.chatbot = Chatbot(

* Running on local URL:  http://127.0.0.1:7869
* Running on public URL: https://d195d7811a2d012aa2.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades,
run `gradio deploy` from the terminal in the working directory to deploy to
Hugging Face Spaces (https://huggingface.co/spaces)

<IPython.core.display.HTML object>
```

[26]: