Python Code Documentation

Your Name

November 4, 2024

1 Introduction

This document provides an overview of the Python code for a Multi-PDF Content-Based Question Answering System. Each line of code is explained for clarity.

2 Code Explanation

2.1 Importing Libraries

```python
import os
import shutil
import streamlit as st
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
import google.generativeai as genai
from langchain_community.vectorstores import FAISS
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.chains.question_answering import load_qa_chain
from langchain.prompts import PromptTemplate
from dotenv import load_dotenv
import pickle
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import hashlib
```

This section imports the necessary libraries for the application. - `os`: Provides a way to use operating system-dependent functionality. - `shutil`: Used for file operations. - `streamlit`: Framework for creating web applications. - `PdfReader`: To read PDF files. - `RecursiveCharacterTextSplitter`: Used to split text into manageable chunks. - `google.generativeai`: API for Google Generative AI. - `FAISS`: A library for efficient similarity search and clustering of dense vectors. - `load_dotenv`: To load environment variables from a .env file. - `pickle`: For object serialization. - `re`: Regular expressions for string processing. - `numpy`: For numerical operations. - `TfidfVectorizer`: For converting a collection of raw documents to a matrix of TF-IDF features. - `cosine_similarity`: Function to compute the cosine similarity between two vectors. - `hashlib`: For hashing passwords securely.

2.2 Loading Environment Variables

```
load_dotenv()
api_key = os.getenv("GOOGLE_API_KEY")
```

This section loads environment variables using the `dotenv` package. It retrieves the Google API key from the environment.

2.3 Configuring Google Generative AI

```
genai.configure(api_key=api_key)
```

Here, the Google Generative AI is configured with the API key.

2.4 Password Hashing Function

```
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
```

This function hashes a password using SHA-256 for secure storage. SHA-256: An acronym for ”Secure Hash Algorithm 256-bit.” It is a cryptographic hash function that produces a 256-bit (32-byte) hash value. SHA-256 is commonly used in various security applications and protocols, including SSL/TLS and blockchain technology. It is known for its security and resistance to collisions (two different inputs producing the same hash).

2.5 User Dictionary

```
users = {
    "user1": hash_password("password1"),
    "user2": hash_password("password2"),
}
```

This dictionary stores usernames and their corresponding hashed passwords for authentication.

2.6 User Login System

```
def login():
    st.sidebar.title("Login")
    username = st.sidebar.text_input("Username")
    password = st.sidebar.text_input("Password", type="password")
    if st.sidebar.button("Login"):
        hashed_password = hash_password(password)
        if username in users and users[username] == hashed_password
            :
```

```
            st.session_state[’logged_in’] = True
            st.session_state[’username’] = username
            st.sidebar.success(f"Welcome␣{username}!")
        else:
            st.sidebar.error("Incorrect␣username␣or␣password")
```

The `login` function provides a login interface on the sidebar and validates the user credentials.

2.7 Logout System

```
def logout():
    if ’logged_in’ in st.session_state and st.session_state[’
        logged_in’]:
        st.sidebar.write(f"Logged␣in␣as␣{st.session_state[’username
            ’]}")
        if st.sidebar.button("Logout"):
            st.session_state[’logged_in’] = False
            st.session_state[’username’] = None
```

This function allows the user to log out of the application.

2.8 Extracting Text from PDFs

```
def get_pdf_text_with_pages(pdf_paths):
    text_chunks_with_pages = []
    for pdf_path in pdf_paths:
        try:
            pdf_reader = PdfReader(pdf_path)
            doc_name = os.path.basename(pdf_path)
            for page_number, page in enumerate(pdf_reader.pages,
                start=1):
                text = page.extract_text()
                if text:
                    text_chunks_with_pages.append((text,
                        page_number, doc_name))
        except FileNotFoundError as e:
            print(f"Error:␣{e}")
    return text_chunks_with_pages
```

This function extracts text from multiple PDF files and keeps track of the page number and document name.

2.9 Text Chunking

```
def get_text_chunks_with_pages(text_chunks_with_pages):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=400,
        chunk_overlap=50)
    chunks_with_pages = []
    for text, page_number, doc_name in text_chunks_with_pages:
        chunks = text_splitter.split_text(text)
        for chunk in chunks:
            chunks_with_pages.append((chunk, page_number, doc_name)
                )
    return chunks_with_pages
```

This function splits the extracted text into smaller chunks for better processing.

2.10 Creating a Vector Store

```
def get_vector_store_with_pages(text_chunks_with_pages, index_name)
    :
    embeddings = GoogleGenerativeAIEmbeddings(model="models/text-
        embedding-004")
    texts, page_numbers, doc_names = zip(*text_chunks_with_pages)
    vector_store = FAISS.from_texts(texts, embedding=embeddings)
    vector_store.save_local(index_name)
    with open("page_numbers_docs.pkl", "wb") as f:
        pickle.dump((page_numbers, doc_names), f)
```

This function creates a vector store from the text chunks and saves it locally for future queries.

2.11 Conversational Chain

```
def get_conversational_chain():
    prompt_template = """
        Answer the question in a detailed and structured way using
            bullet points to ensure clarity and easy understanding.
        Do not provide the answer in paragraph form. Use numbered
            lists for sub-points if applicable.
        If the answer is not available in the provided context,
            simply state: "The answer is not available in the
            context."

        Context:\n {context}\n
        Question: \n{question}\n

        Answer:
    """
    model = ChatGoogleGenerativeAI(model="gemini-1.5-flash",
        temperature=0.1)
    prompt = PromptTemplate(template=prompt_template,
        input_variables=["context", "question"])
    chain = load_qa_chain(model, chain_type="stuff", prompt=prompt)
    return chain
```

This function sets up the conversational chain, which generates answers to questions based on the context.

2.12 User Input and Response Handling

```
def user_input_with_page(user_question, index_name):
    embeddings = GoogleGenerativeAIEmbeddings(model="models/text-
        embedding-004")

    new_db = FAISS.load_local(index_name, embeddings,
        allow_dangerous_deserialization=True)
    search_results = new_db.similarity_search(user_question,
        return_scores=False)

    chain = get_conversational_chain()

    response = chain.invoke(
        {"input_documents": search_results, "question":
            user_question},
        return_only_outputs=True
    )

    return response["output_text"]
```

This function takes user questions and retrieves relevant answers from the vector store.

2.13 TF-IDF Vector Calculation

```
def calculate_tfidf_vector(texts):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(texts)
    return tfidf_matrix
```

This function computes the TF-IDF vectors from the provided texts.

2.14 Cosine Similarity Calculation

```
def calculate_cosine_similarity(vector_matrix):
    cosine_similarities = cosine_similarity(vector_matrix)
    return cosine_similarities
```

This function calculates the cosine similarities between TF-IDF vectors.

2.15 Streamlit Application Layout

```
def main():
    st.title("Multi-PDF␣Content-Based␣QA␣System")

    if ’logged_in’ not in st.session_state or not st.session_state[
        ’logged_in’]:
        login()
    else:
        logout()

        uploaded_files = st.file_uploader("Upload␣PDF␣files",
            accept_multiple_files=True, type=’pdf’)

        if st.button("Process␣PDFs"):
            pdf_paths = []
            for uploaded_file in uploaded_files:
                pdf_path = os.path.join("tempDir", uploaded_file.
                    name)
                with open(pdf_path, "wb") as f:
                    f.write(uploaded_file.getbuffer())
                pdf_paths.append(pdf_path)

            text_with_pages = get_pdf_text_with_pages(pdf_paths)
            chunks_with_pages = get_text_chunks_with_pages(
                text_with_pages)
            index_name = "my_vector_store"
            get_vector_store_with_pages(chunks_with_pages,
                index_name)

        user_question = st.text_input("Ask␣a␣question␣about␣the␣
            PDFs:")
        if st.button("Get␣Answer"):
            if user_question:
                answer = user_input_with_page(user_question,
                    index_name)
                st.write(f"Answer:␣{answer}")
```

This is the main function of the Streamlit app, managing user login, PDF uploads, and question handling.