LangChain Chat with Your Data: Vector stores and Embedding

This document explains the **Retrieval Augmented Generation (RAG)** process using LangChain and OpenAI for embedding, storing, and performing similarity-based retrieval on document chunks. Below are the steps, starting from document loading to edge case handling.

Load Environment Variables:

```
# Define your OpenAI API key
OPENAI_API_KEY = "Replace with your actual API key"
USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"
```

```
import os
import openai
from dotenv import load_dotenv, find_dotenv

# Load environment variables for OpenAI API
load_dotenv(find_dotenv())
openai.api_key = os.getenv("OPENAI_API_KEY")
```

RAG Workflow Overview:

1. Load Documents
2. Split Documents into Chunks
3. Embed Each Chunk
4. Store Embeddings in a Vector Store
5. Perform Similarity Search
6. Handle Edge Cases in Similarity Search

Step 1: Load PDF Documents:

The first step is to load the documents (e.g., PDF files) from your local file system. We use PyPDFLoader from the langchain_community.document_loaders library to extract text content from PDF files.

Vectorstores_and_Embedding.py > ...

```python

from langchain_community.document_loaders import PyPDFLoader # type: ignore

pdf_loaders = [
    # Duplicate documents on purpose - messy data
    PyPDFLoader("docs/cs229_lectures/sfbu-2024-2025-university-catalog-8-20-2024_1.pdf"),
    PyPDFLoader("docs/cs229_lectures/sfbu-2024-2025-university-catalog-8-20-2024_1.pdf"),
    PyPDFLoader("docs/cs229_lectures/sfbu-2024-2025-university-catalog-8-20-2024_2.pdf"),
    PyPDFLoader("docs/cs229_lectures/sfbu-2024-2025-university-catalog-8-20-2024_3.pdf")
]

# Load all documents into `docs` list
docs = []
for loader in pdf_loaders:
    docs.extend(loader.load())
```

Step 2: Split Documents into Chunks:

Since documents may be lengthy, they need to be split into smaller, semantically meaningful chunks to make them manageable for embedding and similarity search. We use RecursiveCharacterTextSplitter to split the documents into chunks.

Vectorstores_and_Embedding.py > ...

```python
# ## Step 2: Split Documents into Chunks
# Use `RecursiveCharacterTextSplitter` to divide each document into smaller, semantically meaningful chunks for embedding.

from langchain.text_splitter import RecursiveCharacterTextSplitter # type: ignore

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 1500,       # Size of each chunk
    chunk_overlap = 150      # Overlap between chunks to retain context
)

splits = text_splitter.split_documents(docs)

print(f"Total number of chunks: {len(splits)}")
```

Output:

```
(venv) vaishnavi@DESKTOP-9V8KJG2:/mnt/c/Users/Mohit/Desktop/Gen AI/Week 7$ python3 Vectorstores_and_Embedding.py
Total number of chunks: 1740
```

Step 3: Embed Each Chunk:

Each document chunk needs to be embedded into vector space. Here, we use OpenAI's embeddings to convert the chunks into vector representations. The vector embeddings help perform similarity-based search.

Vectorstores_and_Embedding.py > ...

```python
# ## Step 3: Embed Each Chunk
# Embed each document chunk to create vectors for similarity search.

from langchain_openai import OpenAIEmbeddings # type: ignore

embedding = OpenAIEmbeddings()

# Example sentences to check similarity
sentence1 = "i like dogs"
sentence2 = "i like canines"
sentence3 = "the weather is ugly outside"

# Embed sentences and compute similarity using dot product
embedding1 = embedding.embed_query(sentence1)
embedding2 = embedding.embed_query(sentence2)
embedding3 = embedding.embed_query(sentence3)

import numpy as np
print("Similarity between sentence1 and sentence2:", np.dot(embedding1, embedding2))
print("Similarity between sentence1 and sentence3:", np.dot(embedding1, embedding3))
print("Similarity between sentence2 and sentence3:", np.dot(embedding2, embedding3))

```

Output:

```
(venv) vaishnavi@DESKTOP-9V8KJG2:/mnt/c/Users/Mohit/Desktop/Gen AI/Week 7$ python3 Vectorstores_and_Embedding.py
Total number of chunks: 1740
Similarity between sentence1 and sentence2: 0.9630397143104906
Similarity between sentence1 and sentence3: 0.7702742223497947
Similarity between sentence2 and sentence3: 0.7590147808716895
```

Step 4: Store Embeddings in a Vector Store:

Once each document chunk is embedded, the next step is to store these embeddings in a vector store (Chroma) for efficient similarity retrieval. The Chroma vector store provides a persistent database for storing vectors and their associated

metadata.

Vectorstores_and_Embedding.py > ...

```python
# ## Step 4: Store Embeddings in a Vector Store
# Set up a vector store (Chroma) to store embeddings for document retrieval.

# Install chromadb package if not already installed
# !pip install chromadb

from langchain_community.vectorstores import Chroma # type: ignore

# Set directory for persistent storage of vector database
persist_directory = 'docs/chroma/'

# Remove old database files if present to avoid conflicts
os.system('rm -rf ./docs/chroma')

# Create vector store from document chunks
vectordb = Chroma.from_documents(
    documents = splits,
    embedding = embedding,
    persist_directory = persist_directory
)

print("\nTotal documents in vector store:", vectordb._collection.count())

```

Output:

```
Total documents in vector store: 1740
```

Step 5: Perform Similarity Search:

With the embeddings stored in the vector store, we can now perform a similarity search to retrieve the most relevant documents based on a query.

Vectorstores_and_Embedding.py > ...

```python
# ## Step 5: Perform Similarity Search
# Run a similarity search using a sample question to find relevant documents.

question = "is there an email I can ask for help"
docs = vectordb.similarity_search(question, k=3)

print("Number of documents retrieved:", len(docs))
print("\nContent of first retrieved document:", docs[0].page_content)

```

Output:

```
Number of documents retrieved: 3

Content of first retrieved document: Library users can find help by using Ask-a-Librarian on the
 the library
catalog, library patrons have two options:
1) Using the computer in the library lobby whose home page is the catalog
2) Access the catalog from the library's website
To access the library's electronic collection, library users have three options:
1) Using the computer in the library lobby
2) Access the e-library via the link on the student/faculty portal:
a. Go to: https://my.sfbu.edu/
b. Click e-Services tab, top right
c. Select eLibrary > ProQuest or O'Reilly
3) 24/7 access from anywhere is provided via EZProxy:
a. Go to: https://elib.sfbu.edu/login
b. Enter your on-campus computer login information
c. Click on "ProQuest Digital Library" or "O'Reilly for Higher Education
MySFBU portal for Faculty and Students
Faculty members use the Canvas LMS and MySFBU faculty portal as tools to help them manage their
c
ourses online, including maintaining their students' academic and attendance records, posting an
updating course syllabi, assignments, instructions, and handout materials. Teaching Assistants
access the system to post homework-related information and useful learning materials for individ
```

Step 6: Edge Case Handling in Similarity Search:

There are edge cases to consider when performing similarity search, such as handling duplicate retrievals or documents that are not directly relevant. Here's how to handle these edge cases:

1. **Duplicate Chunks:** If the dataset contains similar or identical documents, the system might return duplicate results.

2. **Specificity Issues:** Sometimes the retrieved documents may not be specific enough to the query.

Vectorstores_and_Embedding.py > ...

```python
# ## Step 6: Edge Case Handling in Similarity Search
# Simulate edge cases in similarity search, such as duplicate retrievals and specificity issues.

# Case 1: Duplicate chunks from similar documents
question = "what did they say about matlab?"
docs = vectordb.similarity_search(question, k=5)
print("\nContent of first duplicate document:", docs[0])
print("\nContent of second duplicate document:", docs[1])

# Case 2: Specificity issues - retrieves documents not directly relevant to the query
question = "what did they say about regression in the third lecture?"
docs = vectordb.similarity_search(question, k=5)

print("\nMetadata of retrieved documents for specificity test:")
for doc in docs:
    print(doc.metadata)

print("\nContent of least specific document:", docs[4].page_content)
```

Output:

```
Content of first duplicate document: page_content='such as n-grams, Hidden Markov Models, text classifiers, and recurrent neural networks.
Practical assignments and projects allow students to apply their knowledge to real-world' metadata={'page': 153, 'source': 'docs/cs229_lectures/sfbu-2024-2025
-university-catalog-8-20-2024_1.pdf'}

Content of second duplicate document: page_content='such as n-grams, Hidden Markov Models, text classifiers, and recurrent neural networks.
Practical assignments and projects allow students to apply their knowledge to real-world' metadata={'page': 153, 'source': 'docs/cs229_lectures/sfbu-2024-2025
-university-catalog-8-20-2024_3.pdf'}

Metadata of retrieved documents for specificity test:
{'page': 130, 'source': 'docs/cs229_lectures/sfbu-2024-2025-university-catalog-8-20-2024_1.pdf'}
{'page': 130, 'source': 'docs/cs229_lectures/sfbu-2024-2025-university-catalog-8-20-2024_3.pdf'}
{'page': 130, 'source': 'docs/cs229_lectures/sfbu-2024-2025-university-catalog-8-20-2024_1.pdf'}
{'page': 130, 'source': 'docs/cs229_lectures/sfbu-2024-2025-university-catalog-8-20-2024_2.pdf'}
{'page': 129, 'source': 'docs/cs229_lectures/sfbu-2024-2025-university-catalog-8-20-2024_2.pdf'}

Content of least specific document: to achieve a specific purpose with clarity and depth.
Prerequisite: ENGL 101
Mathematics
MATH201 Calculus – I (3 credit hours)
This course is the first of a series in calculus designed for students to build up the
fundamental background of calculus and to learn its applications to very basic problems.
Topics include functions, limits, continuous functions, derivatives and applications,
antiderivatives, composite functions and chain rule, graphing techniques using derivatives,
implicit differentiation, finite integrals, and fundamental theorems of calculus.
(GE – in Mathematics area)
Prerequisite: Pre-calculus subjects
MATH202 Calculus – II (3 credit hours)
This course is the second of the calculus series designed for students to understand
integration techniques and extend the differentiation notion and methods to functions of
multiple variables. Topics include logarithmic and exponential functions and their
derivatives, inverse trigonometric functions, and derivatives, L'Hopital's rule, integration
techniques and their applications, sequence, series, partial derivatives, and improper
integrals.
Prerequisite: MATH201
```

Conclusion:

This document outlines the steps of the Retrieval Augmented Generation (RAG) workflow, from loading documents, splitting them into chunks, embedding them, storing them in a vector store, performing similarity search, and handling edge

cases. This process enables an efficient way of augmenting generative models with retrieval capabilities to improve the accuracy and relevance of answers based on document content.

GitHub URL:

https://github.com/vaishnavi477/Machine-Learning/tree/main/LangChain%20Chat%20with%20your%20Data/Vectorstores%20and%20Embedding

Google Slides:

https://docs.google.com/presentation/d/1ievLnqYwqZY2QzI80fN-NoHBvppPI8ujT3sTykUMUdI/edit?usp=sharing