NATURAL LANGUAGE PROCESSING
AI 829

IIITB

Disease Diagnosis.

—

Mandate 2

Brief Description

Update on project till now:

We have started building the project as per the suggestion given by Sir in the last mandate. Our plan is to create our project using RAG (Retrieval Augmented Generation). We have used

- OpenAIEmbeddings: Utilized to generate vector embeddings for text data using OpenAI's language models.
- OpenAI: Integrated for leveraging advanced natural language processing capabilities, such as text generation and semantic understanding, through AI-powered models.
- langchain: Used for managing language data processing pipelines and vector representations.
- ctransformers: Utilized for integrating transformer-based models into the language processing pipeline.
- sentence-transformers: Employed for generating dense vector representations of sentences using transformer-based models.
- chromadb: Utilized for storing and retrieving document embeddings efficiently.
- nltk: Used for natural language processing tasks such as tokenization, stemming, and stopwords removal.
- spacy: Utilized for advanced natural language processing tasks such as lemmatization, named entity recognition, and part-of-speech tagging.
- PyPDF2: Employed for extracting text data from PDF documents for further processing in the pipeline.

Up to now, we have created the basic architecture of our project. Firstly, we have completed Document Loading and Preprocessing. In this phase, we load medical documents from a specified directory and preprocess each document using the defined preprocessing function.

Next, we have implemented Chunking Documents, wherein the preprocessed documents are split into smaller chunks to facilitate further processing.

Following that, we have performed vector embedding of the text from the PDF file. This provides us with the advantage of storing similar words near each other.

Subsequently, we have stored the embedded data in ChromaDB, successfully creating the vector DB.

Afterward, we have developed the User Query Prompt, allowing users to input a medical query. We embed the query and perform a similarity search in the database. If we find a matching result, we will print it and also cite the source of the data.

Lexical Processing

In lexical analysis, we analyze the structure of words. This is also known as morphology. We look at the root of the word, the affixes, and the word shape. We also look at how the word is used in context. This can help us to understand the meaning of the word.

Lexical analysis is the process of breaking down a piece of text into its individual words. This can be done by hand, or with the help of a computer program.

There are a few different ways to perform lexical analysis. The most basic way is to simply break the text down into a list of words, called a token list. This can be done by splitting the text at every space, and then removing any punctuation marks.

A more sophisticated approach is to use a part-of-speech tagger. This is a program that will assign a part-of-speech tag to each word in the text. This can be helpful for understanding the meaning of the text, as certain words can only be used in certain ways.

Another approach is to lemmatize the text. This means reducing each word to its base form, so that "cats" would become "cat" and "running" would become "run". This can be helpful for identifying different forms of the same word, such as plurals or verb tenses.

Finally, you can also stem the text. This means cutting off any suffixes or prefixes from each word, so that "cats" would become "cat" and "running" would become "run". Stemming can be helpful for dealing with different forms of the same word, but it can also create non-words, so it's not always appropriate.

Tokenization

Tokenization is a foundational concept in natural language processing (NLP) and holds a crucial role in our project, which revolves around processing medical documents and generating responses to medical queries. It involves breaking down an input text into smaller, meaningful units known as tokens. These tokens encapsulate a wide range of linguistic components, including words, terms, sentences, symbols, and other significant elements present within the text.

In the context of our project, tokenization serves as the initial step in preprocessing the text extracted from PDF documents. As part of this preprocessing stage, our approach involves extracting all text from PDF files and normalizing it by bringing words to their root forms. This normalization process, often achieved through techniques like lemmatization, aims to create a clean and standardized representation of the textual data.

In our Project we are predominantly focusing on Tokenization, Lowercasing, Stopword removal, Lemmatization, Remove punctuation

For pre processing we have done

Tokenization: The process of dividing a text into smaller units, such as words or sentences, called tokens.

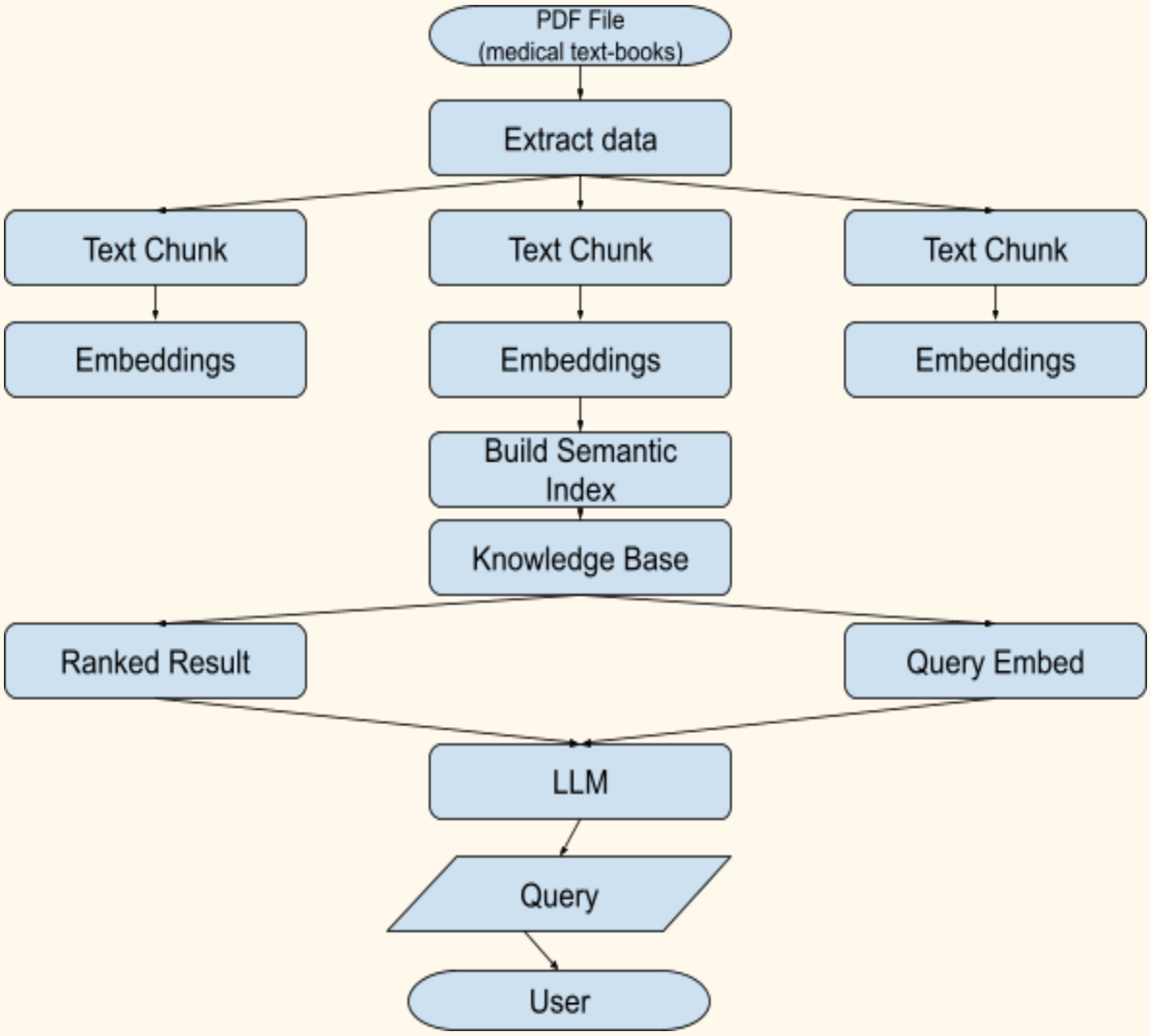
Stopword removal: Eliminating common words that carry little semantic meaning, known as stopwords, from a text.

Lemmatization: Reducing words to their base or dictionary form, called lemmas, to normalize variations like verb conjugations or plural forms.

Punctuation removal: The act of removing punctuation marks from a text to simplify and standardize the data for further analysis or processing.

Work on Feedback from Mandate 1 Evaluation

Sir told to revamp the workflow & don't build everything from scratch, instead use a RAG model to facilitate the project.



1. Extract Data

```
def load_documents():

# loader = PyPDFDirectoryLoader("data/books/")

loader = PyPDFLoader("data/books/symp.pdf")

documents = loader.load()

return documents
```

Documents are loaded from a book, we have multiple books. Will figure out the best best one(s) & use them later on.

2. Pre-processing

```
def preprocess_text(text):

# Tokenization

tokens = re.findall(r'\b\w+\b', text)

# Lowercasing

tokens = [token.lower() for token in tokens]

# Stopword removal

stop_words = set(stopwords.words('english'))

tokens = [token for token in tokens if token not in stop_words]

# Lemmatization

tokens = [token.lemma_ for token in nlp(" ".join(tokens))]

# Remove punctuation

tokens = [token for token in tokens if token not in string.punctuation]

return " ".join(tokens)
```

- Tokenization: It breaks the input text into individual words or tokens. The regular expression r'\b\w+\b' matches whole words.

  Example:

  Input: "Hello, world! How are you?"

  Output: ['Hello', 'world', 'How', 'are', 'you']

- Lowercasing: It converts all tokens to lowercase to ensure uniformity and avoid duplication of words due to case differences.

  Example:

  Input: ['Hello', 'world', 'How', 'are', 'you']

  Output: ['hello', 'world', 'how', 'are', 'you']

- Stopword removal: It removes common words (stopwords) that do not contribute much to the meaning of the text.

  Example:

  Input: ['hello', 'world', 'how', 'are', 'you']

  Output: ['hello', 'world']

- Lemmatization: It reduces words to their base or dictionary form (lemmas) to normalize variations.

  Example:

  Input: ['walking', 'walked']

  Output: ['walk', 'walk']

3. Splitting documents into Chunks

```
def split_text(documents: list[Document]):

# for doc in documents:

# doc.page_content = preprocess_text(doc.page_content)

text_splitter = RecursiveCharacterTextSplitter(

chunk_size=3000,

chunk_overlap=500,

length_function=len,

add_start_index=True,

)

chunks = text_splitter.split_documents(documents)

print(f"Split {len(documents)} documents into {len(chunks)} chunks.")



document = chunks[10]

print(document.page_content)

print(document.metadata)



return chunks
```

Since it's very hard to process all the data at once. The pre-processed data is divided into chunks which are easy to work on.

4. Create Vector Embeddings & Store it in database

```
def save_to_chroma(chunks: list[Document]):

# Clear out the database first.

if os.path.exists(CHROMA_PATH):

shutil.rmtree(CHROMA_PATH)



# Create a new DB from the documents.

db = Chroma.from_documents(

chunks, OpenAIEmbeddings(), persist_directory=CHROMA_PATH

)

db.persist()

print(f"Saved {len(chunks)} chunks to {CHROMA_PATH}.")
```

Embeddings are created from the chunks provided.

> Vector embeddings are a way of representing words, phrases, or entities as numerical vectors in a high-dimensional space. Each dimension in this space corresponds to a feature, and the values in the vector represent the importance or presence of that feature. These embeddings are designed to capture semantic and syntactic relationships between words or entities.

> In other words, when we represent real-world objects and concepts such as images, audio recordings, news articles, user profiles, weather patterns, and political views as vector embeddings, the semantic similarity of these objects and concepts can be quantified by how close they are to each other as points in vector spaces. Vector embedding representations are thus suitable for common machine learning tasks such as clustering, recommendation, and classification.

5. Similarity Search

```
# Search the DB.

results = db.similarity_search_with_relevance_scores(query_text, k=3)

if len(results) == 0 or results[0][1] < 0.7:

print(f"Unable to find matching results.")

return
```

Providing unnecessary extra data to LLMs is also not useful & efficient.

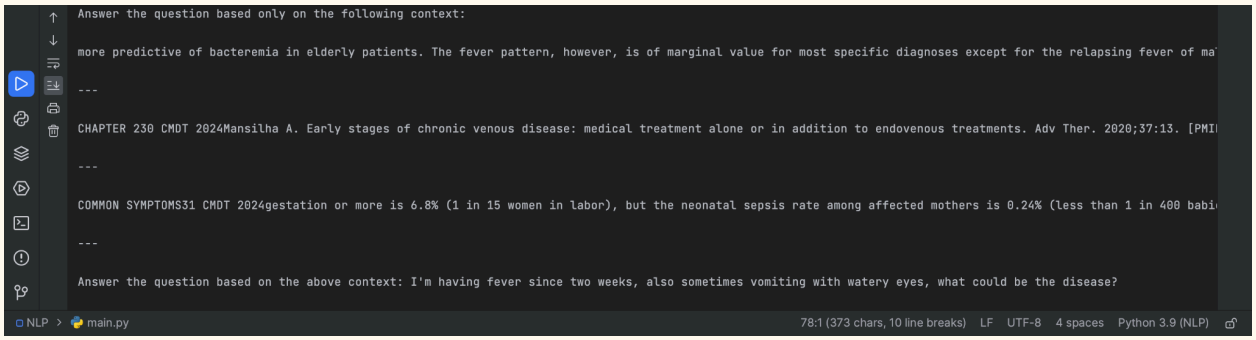
So from the vector embeddings that we created, we do a similarity search based on the vector score they have, like how close they are, and since it's numerically stored it's easy to compare the relevancy of the data with the prompt provided.

6. Creating Prompt

```
PROMPT_TEMPLATE = """

Answer the question based only on the following context:



{context}



---



Answer the question based on the above context: {question}

"""
```

This is where the RAG game execution starts. We mould the prompt in a way that it generates the response based on the context provided, which helps in reducing hallucinations.

Output:



Brief about next Mandate(3)

As data is now prepared which is pre-processed, splitted into multiple chunks. It's the best time to embed any LLM model.

Will figure out which suits this domain well & also if required then fine tune them accordingly.

Also to improve accuracy, can apply reranking & combine multiple models together to yield output.

Project Link: **click here**

Notebook name:NLP_Project1.1.ipynb