Aplicaciones LLM

Técnica RAG: conceptos avanzados

Splitters: conceptos avanzados

- CharacterTextSplitter
  - Utiliza un criterio para separar el texto en fragmentos.
  - Por ejemplo, el salto de línea: “/n”

- RecursiveCharacterTextSplitter
  - Utiliza uno o varios criterios para separar el texto en fragmentos.
  - Por ejemplo, el salto de párrafo y el salto de línea: “/n/n” y “/n”
  - Eso quiere decir que primero separará los fragmentos siguiendo el criterio de salto de párrafo y después, si alguno de los fragmentos es mayor que el límite de tamaño establecido, separará los fragmentos que excedan del tamaño máximo utilizando el segundo criterio (salto de línea).

RetrievalQA Chain: conceptos avanzados

- chain_type = Stuff vs MapReduce:
  - Stuff sencillamente agrega todos resultados de la búsqueda semántica, los convierte en prompt y con ese prompt hace una sola llamada al LLM.
  - El problema de Stuff viene cuando ese resultado agregado supera el límite de la context window.
  - MapReduce hace una llamada al LLM por cada resultado de la búsqueda semántica. Cuando tiene todas las respuestas, hace una nueva llamada al LLM y le pide que teniendo en cuenta todas las respuestas anteriores diseñe una sola respuesta final.
  - El problema de MapReduce es que al hacer más llamadas al LLM es más caro que Stuff.
- return_source_documents = True
  - La respuesta incluirá como metadata las sources utilizadas para confeccionarla.
- chain_type_kwargs = {“prompt”: prompt_template}
  - Para asociar un prompt template que, por ejemplo, diga a chatGPT que responda en base al contexto y que si no encuentra una respuesta en el documento responda “no lo sé”.

Vector database: conceptos avanzados (1)

- Crear la base de datos asociada a un documento privado cada vez que se ejecuta la app es innecesario si el documento privado no ha cambiado. Además, lleva tiempo y coste. Mejor guardarla en un fichero y crear una función que la cree solo al principio.

```
embedding = OpenAIEmbedding()

def create_vector_db():
    loader = CSVLoader(…)
    documents = loader.load()
    vectordb = FAISS.from_documents(documents, embedding)
    vectordb.save_local(“my_vector_database”)

vectordb_file_path = “my_vector_database”
```

Vector database: conceptos avanzados (2)

- Al final del archivo añadimos este código que creará la base de datos si el archivo se está ejecutando como programa principal y no está siendo importado como un módulo en otro script.

```
if __name__ == “__main__”:
    create_vector_db()
```

- Ahora al crear la chain, solo tenemos que recuperar la base de datos guardada:

```
vectordb = FAISS.load_local(vectordb_file_path, embedding)
```

Otros tips interesantes

- langchain.debug = True