UNIVERSIDAD NACIONAL DE ROSARIO

Facultad de ciencias exactas y agrimensura

PROCESAMIENTO DE LENGUAJE NATURAL

Trabajo Práctico n°2

Tomás Rodríguez Griñó
Legajo: R-4643/4

INTRODUCCIÓN

En este documento se encontrará el paso a paso de la resolucion al trabajo práctico número 2 que se dio en la cátedra Procesamiento de lenguaje natural

EJERCICIO 1

El objetivo de este ejercicio es hacer un chatbot que responda consultas sobre un juego de mesa especificado por la cátedra, en mi caso, seria para el juego White Castle. Esto se debe hacer usando la técnica de RAG (Retrieval Augmented Generation)

DESARROLLO DEL PASO A PASO

Lo primero que realicé fue la recolección de datos a partir de diferentes fuentes

- Unas fuente fueron 2 páginas web diferentes en las que hice web scraping
    - https://misutmeeple.com/2023/11/resena-the-white-castle/
    - https://boardgamegeek.com/boardgame/371942/the-white-castle/credits
- Y luego, desde la página de bgg, se descargar diferentes documentos para tener mas datos

Luego de tener todos los datos preparados, los dos web scraping que hicimos, los unimos en un solo texto para después realizar Text splitting, para poder tener cantidades de información más manejables

```
def split_text(texto_completo):
    # Configuración del splitter
    splitter = CharacterTextSplitter(separator="\n", chunk_size=500,
chunk_overlap=50)
    return splitter.split_text(texto_completo)

chunks = split_text(texto_completo)
```

Obviamente todo esto va quedando guardado en la carpeta llamada data

Luego se realizó un proceso de embedding y guardado en ChromaDB

```python
import chromadb
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.core import SimpleDirectoryReader
from chromadb import PersistentClient

# Inicialización de ChromaDB con cliente persistente
client = PersistentClient(path="chroma_data")  # Especifica la ruta donde se guardarán los datos

# Crear o acceder a una colección
collection_name = "embedding_collection"
collection = client.get_or_create_collection(name=collection_name)

# Verificar si la colección se ha creado o accedido correctamente
print(f"Accediendo a la colección: {collection_name}")
```

```python
# Cargar el modelo de embeddings
model_name = "sentence-transformers/all-MiniLM-L6-v2"  # Usamos un modelo preentrenado de Hugging Face
embed_model = HuggingFaceEmbedding(model_name=model_name)

# Cargar documentos desde un directorio
documents = SimpleDirectoryReader(input_dir="data").load_data()

# Extraer textos de los documentos
texts = [doc.text for doc in documents]

# Generar embeddings para cada documento
embeddings = [embed_model.get_text_embedding(doc.text) for doc in documents]

# Verificar el embedding de un documento de ejemplo
print(f"Embedding de ejemplo: {embeddings[0]}")

# Guardar los documentos y sus embeddings en ChromaDB
collection.add(
    documents=texts,  # Los textos de los documentos
    embeddings=embeddings,  # Los embeddings generados
```

```
    metadatas=[{"source": "data"} for _ in documents],  # Metadata
opcional
    ids=[f"doc_{i}" for i in range(len(documents))]  # IDs únicos para los
documentos
)

print("Embeddings guardados exitosamente en ChromaDB.")
```

Luego de hacer esto, podemos realizar una comparación entre un LLM, y un modelo entrenado con ejemplos y embeddings, utilizando los que hicimos anteriormente. En este mismo se pudo observar que el modelo LLM fue mucho más preciso a comparación con el modelo de regresión lineal hecho a partir de ejemplos y embeddings

Luego se decidió hacer el retriever, lo cual es es un componente o sistema diseñado para buscar y recuperar información relevante de un conjunto de datos o documentos en respuesta a una consulta o pregunta. Para el retriever se utiliza la query, para luego obtener el embedding de esta misma, para que contenga la información vectorial y compare con los documentos

Para luego poder realizar el reranking utilizando el MultilingualRankLLM para mejorar la calidad de los resultados de búsqueda o recuperación de documentos. Después de que un sistema inicial (como un retriever) ha recuperado un conjunto de documentos relevantes en respuesta a una consulta, el reranking se encarga de reordenar esos resultados para presentar los más relevantes en la parte superior.