Práctica RAG: Análisis Filosófico y Social de la Gen Z

1. Introducción y Objetivo

El objetivo de esta fase de la práctica fue implementar un sistema **RAG** robusto y programático para analizar un corpus de textos filosóficos académicos y datos de redes sociales utilizando modelos locales llamado Llama-3 a través de LM Studio.

2. Fase de Ingesta y Vectorización (ChromaDB)

La primera etapa fue transformar los documentos crudos (PDFs) en una base de datos vectorial consultable.

A. Pre-procesamiento de Texto (NLP)

Se implementó un script de Python app.py para limpiar el ruido de los documentos originales antes de la indexación.

- **Librerías utilizadas:** nltk, pypdf, langchain.
- **Técnicas aplicadas:**
  - **Normalización:** Conversión a minúsculas.
  - **Limpieza de Stopwords:** Eliminación de palabras vacías (artículos, preposiciones) usando el corpus stopwords de NLTK en español.
  - **Lematización:** Reducción de palabras a su raíz utilizando WordNetLemmatizer para mejorar la coincidencia semántica.
  - **Solución de Errores:** Se resolvió la dependencia de punkt_tab requerida por las nuevas versiones de NLTK.

B. Fragmentación (Chunking)

Para manejar contextos filosóficos largos y complejos, se configuró una estrategia de fragmentación específica:

- **Algoritmo:** RecursiveCharacterTextSplitter.
- **Tamaño del Chunk:** 1000 tokens/caracteres. (Justificación: Preservar argumentos filosóficos completos).
- **Solapamiento (Overlap):** 200 tokens. (Justificación: Mantener la continuidad del contexto entre fragmentos).

C. Generación de Embeddings y Almacenamiento

- **Modelo de Embeddings:** sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2. Se eligió este modelo por su capacidad multilingüe y eficiencia.

- **Vector Store: ChromaDB**. Se implementó en modo persistente (guardado en disco en la carpeta ./chroma_db_proyecto), permitiendo reutilizar los índices sin reprocesar los PDFs.

3. Fase de Conexión y Ejecución (Pipeline RAG)

Una vez creada la base de datos, se desarrolló el script de ejecución run.py para conectar la "memoria" (Chroma) con el "cerebro" (LM Studio).

A. Desafíos de Integración (LangChain)

Se superaron varios desafíos técnicos debido a la modularización reciente de la librería LangChain (v1.2.0+):

1. **Error de Módulos (langchain.chains):** Se identificó que RetrievalQA había sido movido.
2. **Solución:** Se instaló el paquete langchain-classic y langchain completo para restaurar la compatibilidad con las cadenas de interrogación.
3. **Conector LLM:** Se migró de conectores genéricos a langchain-openai (ChatOpenAI) para aprovechar la compatibilidad nativa del servidor local de LM Studio.

B. Configuración del Retriever

- **Tipo de Búsqueda:** Similitud de coseno (similarity).
- **Top-K:** 4. El sistema recupera los 4 fragmentos más relevantes para construir la respuesta.

C. Configuración del Modelo Generativo (LLM)

- **Servidor:** LM Studio Local Server (http://localhost:1234/v1).
- **Temperatura:** 0.3. Se redujo la creatividad para favorecer respuestas más analíticas y fieles al texto fuente.
- **Prompt Engineering:** Se diseñó una plantilla (PromptTemplate) específica instruyendo al modelo a actuar como un "asistente de investigación experto en filosofía", forzando la citación de autores (Foucault, Bauman, etc.) si aparecen en el contexto.

4. Resultados y Flujo de Trabajo Final

El sistema final opera bajo el siguiente flujo:

1. **Entrada:** El usuario hace una pregunta filosófica compleja.
2. **Recuperación:** ChromaDB busca los 4 fragmentos de texto más similares semánticamente en el corpus procesado.
3. **Aumentación:** Se insertan estos fragmentos en el Prompt del sistema.
4. **Generación:** LM Studio recibe el prompt enriquecido y genera una respuesta fundamentada.
5. **Evidencia:** El script devuelve tanto la respuesta generada como los metadatos de las fuentes (nombre del archivo y página) para validación académica.

Comparativa: Script vs. Interfaz Gráfica

Se concluyó que el uso de scripts programáticos (Python + Chroma) es superior a la interfaz gráfica básica ("Chat with Docs") para fines académicos porque permite:

- Control exacto del chunking (vital para textos densos).
- Recuperación de citas precisas (Source Documents).
- Persistencia de la base de datos entre sesiones.

Anexos

app.py

```
import os
import nltk
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from langchain_community.document_loaders import PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import Chroma



def limpiar_texto(texto):
   texto = texto.lower()
   tokens = nltk.word_tokenize(texto)
   stop_words = set(stopwords.words('spanish'))

   tokens = [word for word in tokens if word.isalnum() and word not in
stop_words]

   lemmatizer = WordNetLemmatizer()
   tokens = [lemmatizer.lemmatize(word) for word in tokens]

   return " ".join(tokens)

carpeta_pdfs = "./dataset_rag/"
documentos_procesados = []
```

```
print("Procesando ")
for archivo in os.listdir(carpeta_pdfs):
    if archivo.endswith(".pdf"):
        ruta = os.path.join(carpeta_pdfs, archivo)
        loader = PyPDFLoader(ruta)
        docs = loader.load()

        for doc in docs:
            doc.page_content = limpiar_texto(doc.page_content)
            documentos_procesados.append(doc)

print(f"Procesadas {len(documentos_procesados)} páginas.")

print("\nChunking")

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200
)
chunks = text_splitter.split_documents(documentos_procesados)

print(f"Dividido en {len(chunks)} chunks.")

embedding_model =
HuggingFaceEmbeddings(model_name="sentence-transformers/paraphrase-multili
ngual-MiniLM-L12-v2")

try:
    vector_db = Chroma.from_documents(
        documents=chunks,
        embedding=embedding_model,
        persist_directory="./chroma_db_proyecto"
    )
    print("La base de datos se encuentra en la carpeta: ./chroma_db")
except Exception as e:
    print(f"error al crear el Vector Store: {e}")
```

run.py

```python
import sys
import os

from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate

try:
    from langchain.chains import RetrievalQA
except ImportError:
    try:
        from langchain_classic.chains import RetrievalQA
    except ImportError:
        print("ERROR RetrievalQA'.")
        sys.exit(1)

CARPETA_CHROMA = "./chroma_db_proyecto"
URL_LM_STUDIO = "http://localhost:1234/v1"

if not os.path.exists(CARPETA_CHROMA):
    print(f"Error '{CARPETA_CHROMA}'.")
    sys.exit()

embedding_model =
HuggingFaceEmbeddings(model_name="sentence-transformers/paraphrase-multili
ngual-MiniLM-L12-v2")

print("Conectando conchroma...")
vector_db = Chroma(
    persist_directory=CARPETA_CHROMA,
    embedding_function=embedding_model
)

retriever = vector_db.as_retriever(search_kwargs={"k": 4})

print(f"Conectando con LM Studio: {URL_LM_STUDIO}...")
```

```
llm = ChatOpenAI(
    base_url=URL_LM_STUDIO,
    api_key="lm-studio",
    temperature=0.3,
    model="local-model"
)

template_filosofico = """
Eres un asistente de investigación experto en filosofía.
Utiliza el siguiente contexto para responder a la pregunta.

CONTEXTO:
{context}

PREGUNTA:
{question}

RESPUESTA DETALLADA:
"""

prompt = PromptTemplate(
    template=template_filosofico,
    input_variables=["context", "question"]
)

qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    return_source_documents=True,
    chain_type_kwargs={"prompt": prompt}
)

print("\nEscribe 'bye' para salir.")
print("="*60)

while True:
```

```
pregunta = input("\nHaz tu pregunta: ")
if pregunta.lower() in ['bye']:
    break

try:
    print("Consultando ...")
    resultado = qa_chain.invoke({"query": pregunta})
    print("\nRESPUESTA:")
    print(resultado["result"])
    print("\nFUENTES:")
    for doc in resultado["source_documents"]:
        print(f" - {doc.metadata.get('source', 'Desconocido')}")
except Exception as e:
    print(f"Error: {e}")
```