Trabajo práctico final

Procesamiento del lenguaje natural

Alumno: Max eder

1)

Introducción:

Para el trabajo final de la materia, se requería la construcción de un chatbot experto en un tema a elección, utilizando RAG (Retrieval-Augmented Generation), técnica con la cual, a un modelo de LLM (lenguaje de gran escala), se le brinda información adicional, obtenida de una fuente de conocimiento externo, para que la respuesta a una consulta, sea más precisa. En el caso de este trabajo práctico, se pedía que el contexto, o información adicional, sea extraída de tres fuentes de conocimiento diferentes, pdfs, tabular y bases de datos de grafos. Por medio de un clasificador, se requería perfilar las consultas hacia la base de datos que contenía el tema específico, para sumarle contexto al modelo llm y que este responda.

En mi caso, elegí tema, la segunda guerra mundial. Alimentando una base de datos vectorial, con información extraída de 3 archivos pdfs que suman aproximadamente 250 páginas, la tabular de un archivo .csv y la base de datos de grafos con consultas a wikidata. La idea era que a partir del clasificador, se obtengan los datos desde la base tabular, en caso de que la consulta sea sobre alguna fecha específica de algún evento de la guerra, la base de datos de grafos, si la consulta es sobre algún personaje de esta misma y la base de datos vectorial para consultas más generales que requieran más desarrollo.

Desarrollo:

Pdfs:

En un primer paso, extraje los archivos contenidos en una carpeta de drive con la herramienta *fitz* de *PyMuPDF*, luego realicé el el split con la herramienta *CharacterTextSplitter* de la librería *langchain*, en mi caso, elegí un tamaño de chunk (fragmentos en los que se divide el texto) de 800 caracteres. A estos le realicé una liempieza, eliminando los saltos de linea (/n) y los links a las páginas que figuraban en los archivos.

Una vez teniendo los chunks limpios, usando el modelo *sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2*, realicé los embeddings y los volqué en una base vectorial *chromaDB.*

La función para la consulta a la base de datos vectorial, lo que hace es hacer el embedding de la consulta ingresada, compararlo con los de la base y devolver el chunk con el embedding más cercano.

Tabular:

El código para extraer información del archivo .csv, primero lo que hace es, con la técnica NER de *Spacy*, de la consulta original, se queda con la ubicación o el evento que se desea conocer la fecha. Luego le práctico una traducción a inglés, ya que el archivo esta en este idioma, y una vez que tengo el dato que me interesa, recorro el archivo con pandas y devuelve en forma de contexto, la fila con nombre de evento y la fecha correspondiente.

Base de datos de grafos:

Para este caso, como en el anterior, tuve que preprocesar la consulta. Con la técnica NER, lo que hago es quedarme solo con el nombre del personaje para hacer la query en wikidata, con la librería SPARQLWrapper, me conecto al endpoint de wikidata y busco el elemento que coincida con el nombre extraído, luego traigo la descripción para retornar el contexto.

Clasificadores:

Como exigía el trabajo, hice dos clasificadores, uno basado en LLM y otro con ejemplos y embeddings.

En el caso de LLM, use el modelo joeddav/xlm-roberta-large-xnli, al cual agregué un prompt donde explicaba como tenía que clasificar y sume un ejemplo. Este es un modelo zero shot de clasificación. No obtuve buenos resultados con una serie de ejemplos que utilicé para probar.

Para el otro caso, alimente con ejemplos los tres casos de clasificación, a estos les realice un embedding y la idea era que a la consulta que se ingresa, se le practique un embedding, y por similaridad de coseno, devuelva el grupo mas cercano. Con este método de clasificación, obtuve mejores resultados, por lo cual, lo elegí para continuar el trabajo.

Modelo y chatbot:

Para el modelo LLM, utilicé zephyr de HuggingFace, entre otras cosas, porque es un modelo que funciona muy bien con la tecnica RAG, según habíamos visto con ejemplos brindados por la cátedra y porque es una herramienta gratuita, que lo único que exige es crear una cuenta en HuggingFace.

Para la generación de la respuesta, coloque un prompt en que se le indica al modelo que utilice el contexto brindado por las consultas a las fuentes de conocimientos externas, para que evitar alucinaciones y que se conteste de forma concisa y precisa.

Para el interfaz del chatbot, utilicé la librería *Gradio*, con una función, se integran todas las generadas para los pasos anteriores, entrando con la consulta como argumento y se clasifica, se le suma el contexto y se genera la respuesta obteniendo el siguiente resultado:

Conclusión:

Si bien creo que logré que el chatbot responda de forma dinámica las consultas, basándose en el contexto que se agrega, noto que podría tener algunas fallas en ciertas cuestiones.

Por ejemplo, si se hace una pregunta que se perfila para buscar en los PDF y esta información no se encuentra, el modelo igual va a devolver un chunk

por proximidad que probablemente no tenga relación con la información que se desea obtener.

En el caso de los clasificadores, creo que podría haber hecho algo más robusto, con otra ingeniería de prompts para el caso de Zero Shot, o generar un modelo de clasificación con más ejemplos en el caso de los embeddings.

En definitiva, todo el trabajo fue de una gran complejidad, se exigía en el mismo la utilización de prácticamente todas las técnicas que vimos en las unidades del cursado.

Enlaces a librerias utilizadas:

https://pypi.org/project/PyMuPDF/ (PyMuPDF)

https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/character_text_splitter/ (Character_text_splitter)

https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 (sentence-transformers)

https://docs.trychroma.com/ (chromaDB)

https://huggingface.co/joeddav/xlm-roberta-large-xnli (modelo zero shot)

https://huggingface.co/HuggingFaceH4/zephyr-7b-alpha (modelo LLM)

https://www.gradio.app/ (gradio)

2)

a)

Rerank es una herramienta que mejora la calidad de los resultados de búsqueda reorganizando los documentos recuperados inicialmente para que los más relevantes aparezcan primero. Esto es útil porque asegura que los resultados más importantes estén en la parte superior, minimiza la aparición de resultados no relacionados y ahorra tiempo y recursos. Funciona dividiendo la información en partes más pequeñas y detalladas, utilizando métodos avanzados para mejorar la coincidencia de palabras clave y contexto, y generalizando mejor las nuevas consultas y documentos, mejorando así la calidad de la búsqueda. Este proceso mejora la precisión y la experiencia del usuario al reducir el ruido y proporcionar respuestas más claras y concisas.

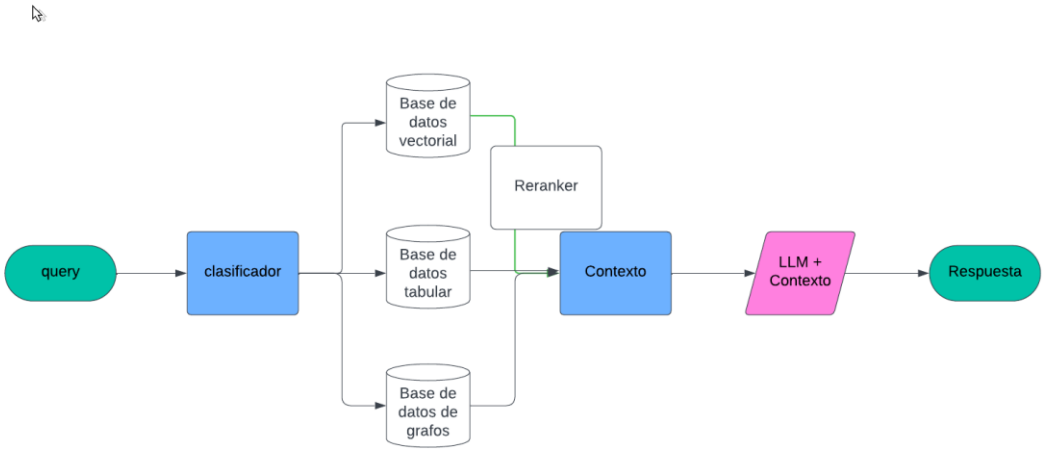
Existen diferentes métodos para hacer el rankeo, como:
Modelos de Encoders Cruzados (Cross-Encoders): Estos modelos utilizan un mecanismo de clasificación para evaluar la similitud entre pares de datos, como una consulta y un documento. A diferencia de los embeddings tradicionales, los cross-encoders operan en pares y calculan un puntaje de similitud que permite una comprensión más detallada de la relación entre la consulta y el documento.

Modelos de Interacción Tardía (Late Interaction Models): Ejemplos como ColBERT adoptan un enfoque en el que la interacción entre la consulta y los documentos se realiza después de que ambos han sido codificados de manera independiente. Esto permite pre-computar las representaciones de los documentos y utilizar técnicas como el pooling para calcular la similitud, lo que reduce la carga computacional y mejora la eficiencia.

Modelos de Lenguaje Grande (Large Language Models, LLMs): Modelos como RankZephyr y RankT5 utilizan grandes modelos de lenguaje para realizar el rerankeo. Estos modelos son capaces de manejar consultas y documentos con alta precisión debido a su capacidad para entender el contexto y las sutilezas del lenguaje.

APIs de Modelos de Lenguaje (LLM APIs): Servicios privados como GPT y Claude ofrecen rerankeo utilizando modelos de lenguaje avanzados a través de una API. Estos servicios suelen proporcionar los mejores resultados en términos de precisión, pero a un costo elevado.

Modelos de Rerankeo Basados en API: Servicios privados como Cohere y Jina proporcionan soluciones especializadas para rerankeo a través de APIs. Estos modelos combinan alta calidad de rendimiento con costos medios, ofreciendo flexibilidad y eficiencia en la reordenación de documentos.



b)

En mi proyecto, aplicaría Rerank después de obtener el contexto, si el clasificador perfila la consulta a la base de datos vectorial. En mi caso devuelve el embedding más cercano, pero si este paso devolvería los n más cercanos se podría utilizar. Esto aseguraría que los resultados más relevantes se utilicen para construir la respuesta, mejorando así la efectividad del chatbot.

Mastering RAG: How to Select A Reranking Model - Galileo (rungalileo.io)