Facultad de
Ciencias Exactas,
Ingeniería y Agrimensura

Trabajo práctico NLP

2° Semestre - Año 2025

Nicolás Mancini

Ejercicio 1: Se crean tres fuentes de datos, que van a ser usadas para responder preguntas del usuario, de acuerdo a los siguientes criterios:

Base Vectorial: almacena todos los documentos procesados como embeddings dentro de una única colección, así tenemos:

- Manuales en formato Markdown
- FAQs en JSON
- Reseñas de usuarios en TXT

Para fragmentar los documentos que componen la base vectorial se utilizó chunk_text_recursive, para lograr respetar la estructura interna del texto. A diferencia de otros métodos que cortan el contenido de forma arbitraria, el enfoque recursivo permite identificar unidades semánticas más coherentes, para garantizar que cada chunk preserve significado propio y no pierda contexto. Por esta razón, se decidió usarlo para obtener mejores resultados en la calidad de los embeddings, facilitando luego, una mayor precisión del clasificador y del modelo de lenguaje al momento de recuperar, interpretar o reconstruir información.

Base de Datos Tabular: Es un DataFrame grande que contiene información previamente procesada y calculada, como métricas, productos, atributos numéricos, relaciones derivadas o información consolidada. Tiene el objetivo de responder cuando la consulta del usuario requiere realizar filtros estructurados, cálculos concretos o análisis estadísticos sobre datos tabulados.

Base de Datos en Neo4j: Es una base de datos orientada a grafos en Neo4j, donde se representan relaciones explícitas entre entidades, obtenidas a partir de los archivos que contienen productos, inventario, ventas históricas, tickets de soporte y devoluciones. Facilita responder consultas que dependen del contexto relacional

Respecto a la tarea de clasificación de intención, se decide utilizar KNN como clasificador, por su simplicidad y eficiencia, con el objetivo de trabajar en conjunto con modelo de embedding. Dado que el modelo de embeddings ya encapsula de manera efectiva la semántica del texto, KNN puede aprovechar plenamente la estructura del espacio embedding y tomar decisiones en función de la distancia entre vectores. De esta forma, se reduce la complejidad del pipeline final, y el riesgo de sobreajuste.

Como LLM, se seleccionó el modelo Gemini 2.0 flash por su capacidad de ofrecer un gran resultado manteniendo un buen rendimiento. Esta versión "Flash", que está diseñada para garantizar una baja latencia y un alto rendimiento de consultas, dio mejores resultados frente a otras opciones previamente usadas como, ollama.

Sin embargo, su principal desventaja es el límite de tokens disponible en el plan gratuito, que restringe la cantidad y complejidad de las interacciones posibles. Aun así, se decidió emplearlo porque facilita la ejecución directamente en la nube sin necesidad de infraestructura local, lo que resultó práctico para el desarrollo.

Finalmente, se combinan todas las funciones para atender las preguntas del usuario. En primer lugar, la memoria conversacional almacena los últimos intercambios entre el usuario y el asistente. Esto permite que el modelo mantenga continuidad entre mensajes. Cada vez que el usuario envía una consulta, la clase de memoria produce un texto formateado que resume el historial reciente, el cual luego se incorpora al prompt del LLM para que la conversación tenga coherencia.

Cuando el usuario hace una pregunta, se ejecuta un pipeline que detecta el tipo de consulta y recupera información relevante desde diferentes fuentes: tablas, vectores o grafos. Dependiendo del tipo de resultado, se convierte la información en un formato comprensible para el modelo, para incorporarlo al prompt y lograr que el modelo pueda usarlo como base de conocimiento al generar la respuesta. Luego se construye un prompt completo que combina cuatro

elementos: las instrucciones del asistente, el historial conversacional, la información recuperada y el mensaje actual del usuario. Una vez generada la respuesta, se guarda la guarda junto con el mensaje del usuario dentro de la memoria conversacional, de modo que influye en los intercambios siguientes. Así, solo ejecutamos un bucle que gestiona la interacción: recibe el mensaje del usuario y muestra la respuesta, permitiendo mantener una conversación fluida y contextualizada.

Ejercicio 2: El código crea un asistente inteligente que utiliza el modelo Gemini como motor principal, y que además tiene la capacidad de usar herramientas externas para obtener información según la necesidad de cada consulta del usuario. Se define:

- Una herramienta de búsqueda documental para recuperar texto de manuales o reseñas mediante un retriever vectorial y un reranker.
- Una herramienta tabular que consulta directamente un DataFrame maestro para obtener precios, stock o datos estructurados.
- Una herramienta de grafos que sirve para responder preguntas sobre relaciones como compatibilidades o recomendaciones
- Una herramienta de analytics que, cuando el usuario lo pide, genera gráficas automáticamente.

Todas estas herramientas se agrupan para que el agente las tenga disponibles. Luego se construye un prompt ReAct personalizado, donde el modelo recibe instrucciones claras sobre cómo debe razonar: antes de responder debe analizar si hace falta usar alguna herramienta, elegir la adecuada basándose en reglas simples, realizar la acción con el formato Thought/Action/Observation, y finalmente generar una respuesta final en español basada en los resultados obtenidos. El prompt también indica cómo estructurar el uso de herramientas y obliga al modelo a ser honesto en caso de que no se encuentre información.

Finalmente se define la última función: recibe una pregunta del usuario, invoca al agente y devuelve la respuesta generada, ya sea con información directa o utilizando alguna herramienta, cuando es posible.