Trabajo Práctico Final - Procesamiento del Lenguaje Natural - TUIA 2024

Mateo Gravi Fiorino.

Ejercicio 1 - RAG.

Crear un chatbot experto en un tema a elección, usando la técnica RAG (Retrieval Augmented Generation). Como fuentes de conocimiento se utilizarán al menos las siguientes fuentes:

- Documentos de texto
- Datos numéricos en formato tabular (por ej., Dataframes, CSV, sqlite, etc.)
- Base de datos de grafos (Online o local)

El tópico elegido es **"Mitología Griega"**.

- Los datos consisten en un PDF sobre mitología griega.
- pdf

- Un archivo csv, generado propio con información en wikipedia.

- Una base de grafos creada con WikiData

Luego de realizar la carga y procesamiento simple de los datos, pasamos a realizar Chunks utilizando la técnica **RecusiveSplitter de langchain**. Esto lo realizamos para el archivo csv y para el archivo pdf.

Luego utilizamos el modelo de **embeddings de openAI** para la creación de los mismos sobre estos documentos.

A continuación seteamos la base de datos de chroma db y creamos un colección. Convertimos los embeddings a listas y los agregamos a nuestra colección.

Modelo.

Definimos un template de zephyr para la parte de modelo de clasificacion basado en cercanías.

Función zephyr_chat_template
Esta función define una plantilla Jinja para renderizar una conversación entre un usuario y un asistente. La función toma una lista de mensajes y un indicador opcional para agregar un prompt de generación.

Parámetros:

messages (List[Dict[str, str]]): Una lista de diccionarios que representan los mensajes en la conversación. Cada diccionario tiene una clave 'role' que puede ser 'user', 'assistant', 'system', etc., y una clave 'content' que contiene el texto del mensaje.
add_generation_prompt (bool, opcional): Un indicador para agregar un prompt de generación al final de la conversación. Por defecto, es True.

Retorno:

Una cadena que representa la conversación renderizada según la plantilla Jinja.

Luego mediante la funcion **answer cleaner** obtenemos la respuesta del modelo

Esta función toma un texto de entrada y busca palabras clave específicas ("Deity" y "Myth"). Si encuentra una coincidencia, extrae la información relevante y la devuelve en una lista. Si no encuentra ninguna coincidencia, devuelve una lista con dos elementos None.

Parámetros:

texto (str): El texto que se va a analizar para encontrar palabras clave y extraer información.
Retorno:

Una lista con dos elementos. El primer elemento es la palabra clave encontrada ("Deity" o "Myth") o None si no se encuentra ninguna palabra clave. El segundo elemento es la información extraída (el nombre de la deidad entre corchetes) o None si no hay información adicional.

Fue una buena aproximación para luego implementar un modelo de QA con Rag.

Enlaces útiles:

- Modelo zephyr
- Modelo embeddings

Modelo basado en LLM.

Ya que tengo acceso a una API de openAI, decidí utilizarla también para esta parte del trabajo. Se implementa el modelo de chatgpt-3.5.

Se importan diversas librerías necesarias para la manipulación de datos (pandas), visualización (matplotlib), procesamiento de lenguaje natural (spacy), manejo de grafos (networkx), y el cliente de OpenAI (openai).

Se carga el modelo de Spacy para el procesamiento de lenguaje natural en inglés.

Función get_embedding:
Obtiene el embedding de un texto dado usando el modelo especificado. El embedding es una representación numérica del texto en un espacio de alta dimensión.
Función strings_ranked_by_relatedness:
Recupera textos relevantes de ChromaDB basados en la consulta usando el embedding de la consulta.

Función num_tokens:
Devuelve el número de tokens en una cadena de texto, lo cual es útil para controlar el límite de tokens en una solicitud a OpenAI.

Función extract_relevant_nodes_and_edges:
Extrae nodos y relaciones relevantes del grafo basados en la consulta.

Función query_graph_database:
Genera una lista de textos relevantes basados en la consulta y el grafo local.

Función query_message
Genera un mensaje para GPT con textos relevantes de ChromaDB y el grafo local, respetando un límite de tokens (token_budget).

Función ask
Responde a una consulta usando GPT, una colección de Chroma y datos relevantes del grafo local. Genera el mensaje, lo envía a OpenAI y devuelve la respuesta.

Ejemplo de uso:

```
query = "¿Quién es Zeus en la mitología griega?"
response = ask(query, collection, graph)
print(response)
```

Ejercicio 2:

1.
Rerank es como por ejemplo, cuando haces una búsqueda en Google y se obtiene una lista de páginas que podrían responder a la pregunta. Pero la primera lista que ves no siempre está en el orden que mejor responde tu pregunta. Rerank es tomar esa lista inicial y reorganizarla para que las páginas más útiles salgan primero.

Cómo Funciona el Rerank:

Primera Búsqueda: Imaginemos que estás buscando recetas de pasta en internet. Haces una búsqueda y tenes un montón de recetas. La primera lista que ves puede tener recetas que no son las mejores o más relevantes.

Primera Clasificación: Los resultados iniciales están ahí, pero no todos están ordenados como ideal. Puede que las recetas más útiles estén al final y las menos interesantes al principio.

Rerank: Acá es donde entra el rerank. Tomas la lista de recetas que encontraste y la volves a ordenar de manera más inteligente. Podes usar un sistema que sepa cuál receta es más popular, más recomendada, o que se ajuste mejor a lo que estás buscando.

Resultado Final: Al final, obtenes una lista de recetas ordenada para que las más útiles y relevantes estén al principio. Así cuando miras los resultados, es más fácil encontrar la receta perfecta sin tener que revisar toda la lista.

Diagrama Simple

Copiar código
[Inicio]
  |
  v
[Primera Búsqueda]
  |
  v
[Lista Inicial de Resultados] -- (Reordenar) --> [Rerank]
  |
  v
[Lista Final de Resultados]
  |
  v
[Resultados Útiles]

<u>En el diagrama:</u>

Primera Búsqueda: Encuentras una lista de resultados.
Lista Inicial de Resultados: Los resultados que encontraste, aunque no estén en el mejor orden.
Rerank: Reorganizas los resultados para que los más útiles estén arriba.
Lista Final de Resultados: Los resultados ajustados y ordenados de mejor a peor.
Resultados Útiles: La información que realmente te ayuda.

2. Aplicación del Rerank en el Código

En el chatbot RAG, podría usar rerank para mejorar los resultados que se obtiene de la base de datos ChromaDB antes de que el modelo GPT genere la respuesta.