Trabajo Práctico Final

Procesamiento del Lenguaje Natural

FCEIA

Docentes:

- Manson, Juan Pablo
- Geary, Alan
- Leon Cavallo, Andre Carolina
- D’ Alessandro, Ariel

Alumno:

- Petetta, Nicola

Idea

Rag-Chatbot que está especializado en astronomía, se espera que pueda proporcionar al usuario una herramienta extra cuando tenga dudas sobre el tema, haciendo que sea más accesible a todos los que deseen aprender sobre el tema.

Esta idea no solo cuenta con un modelo pre entrenado con información sobre astronomía sino que también obtendrá información de varios tipos de varios lugares, en este caso se usó una base de datos vectorial para guardar información de libros de astronomía los cuales fueron procesados y fragmentados antes de guardar los embeddings de estos archivos.

Se usó un archivo CSV con información del Sol extraída de wikipedia, el cual se transformó en string para que el modelo pueda usar esta información.

Y por último se consultó a una base de datos de grafos de código abierto DBpedia, para traer la información básica de cada planeta del sistema solar.

Esta información una vez obtenida funciona para darle mayor contexto al modelo así genera respuestas más informadas.

Breve explicacion del codigo

Importar librerías y archivos

Primero se instalan las librerías que van a ser usadas, después se importan.

Se procede a dar una url a una carpeta de google drive donde están todos los archivos que van a ser usados.

Read_pdf: es una función que toma la ruta de un pdf como argumento, usando Fitz lo abre y extrae el texto de todas sus páginas, después cierra el archivo y devuelve el string de texto con todo el contenido del pdf.

Procesado de Texto

Normalize_text: toma texto como entrada y procede a normalizar, anteriormente esta función hacía stemming y otras operaciones como remover acentos, pero se cambió para solo eliminar las stop words y tokenizar el texto. Esta función usa la biblioteca NLTK.

Process_and_split_pdfs: esta función usa las dos anteriores para leer y normalizar el texto, después con la librería LangChain y el método RecursiveCharacterTextSplitter se procede a hacer chunks de los archivos pdf que se proveen como una lista a esta función.

Para leer el CSV que vamos a usar que tiene información del sol, tomada de wikipedia, con pandas lo leemos.

La función buscar_informacion_dataframe: esta función finalmente no se usa, pero la idea era que sirva para buscar información dentro del dataframe, mediante el nombre de la sección que esté buscando, iba a filtrar la fila para obtener el valor que sería la descripción. Y toma el primer elemento. Finalmente esto se cambio por df_sol.to_string() en la función dar_contexto.

BDD Vectorial

Ahora se pasa a hacer la bdd vectorial, se carga un modelo de embeddings con la librería tensorflowhub.

Con la librería chromadb se hace un base de datos llamada astronomía basics

Y una función store que toma un diccionario con esta información sobre astronomía y también toma la colección creada de chromadb donde se van a almacenar los documentos y sus embeddings.

BDD de grafos

Extraer_consulta_bdd_grafos: lo que hace es preparar una consulta con la librería spaCy, buscando entidades en la consulta, si reconoce una toma la primera y la guarda en una variable, la cual se pasa a la siguiente función

Consultar_bdd_grafos: arma una consulta SPARQL con el filtro que busca la entidad pasada por la consulta, con request hace una solicitud a DBpedia y la respuesta se procesa como un JSON y se extrae la información de la primera entidad que encuentre, en caso de que no encuentre datos para la entidad o haya un error en la solicitud se muestran estos respectivos mensajes de error.

En nuestro caso lo que hace es consultar a DBpedia para obtener información de planetas y la información devuelve el nombre del respectivo planeta y un resumen de la información sobre este

Determinación de fuente

Mediante un diccionario de ejemplo de few shots para el modelo

La función determine_source toma una pregunta del usuario y este diccionario, estos argumentos se convierten en un embedding usando SentenceTransformer('paraphrase-MiniLM-L6-v2')

Después se calcula la similitud del coseno con la pregunta del usuario y cada uno de los ejemplos proporcionados para determinar cuál fuente está más “cerca” de la pregunta del usuario.

Generar Respuesta

Solo se usa la función generate_answer que llama a la api de hugging face y genera texto de un prompt dado, los parámetros temperatura, top-k y top-p son los que determinan la longitud y que tan “creativo” va a ser la generación de text.

Plantilla de texto

Zephyr_chat_template: Utiliza el motor de plantillas Jinja2 para generar una representación de texto a partir de un conjunto de mensajes en un formato específico toma una lista de mensajes (cada mensaje con un contenido y un rol) y utiliza una cadena de plantilla Jinja2 para generar una representación de texto de la conversación. Puede manejar diferentes roles y añadir un prompt de generación si se especifica.

Preparar Prompt

Prepare_prompt: construye un prompt que incluye información de contexto y una pregunta específica. El prompt se estructura en base a un formato predeterminado y se utiliza para obtener respuestas del modelo

Dar_contexto

Toma una consulta y una fuente determinada de la función anterior, dependiendo del valor de la fuente que puede ser vector de la bdd vectorial, dataframe del dataset o graph-db si la fuente se determinó de la bdd de grafos.
A context_info se le va a asignar la información respectiva dependiendo de la fuente determinada.

Process_and_generate_answer

Esta es la función que se encarga de procesar las consultas, determinar la fuente de contexto, obtener el contexto y generar una respuesta basadas en un prompt que depende de cada fuente de contexto

Interfaz

Esta interfaz proporciona una manera sencilla para que los usuarios hagan preguntas sobre astronomía y reciban respuestas generadas por el modelo basadas en la información disponible en diferentes fuentes.

Datos interesantes