TUIA

TP Final

Materia: Procesamiento de Lenguaje Natural

Alumno: Pablo Pistarelli

Legajo: P-5195/1

Ejercicio 1:

➢ Introducción:

Se solicitó crear un chatbot experto usando la técnica RAG sobre un tema a elección y en mi caso elegí Básquetbol.

Para documentos de texto utilicé distintos archivos sobre el reglamento de básquetbol y se almacenaron en una base de datos vectorial ChromaDB; para la base de datos tabular utilicé una base SQLite con información sobre los resultados de los distintos países/selecciones en los mundiales de este deporte; para el resto de las preguntas que no pertenecen a estas categorías utilicé la base de grafos de Wikidata.

➢ Desarrollo detallado de los pasos para la solución del problema, y justificaciones correspondientes:

Archivos Pdf e información tabular:

Los archivos pdf y el script sql para llenar la base de datos tabular se subieron en un drive compartido y se ejecuta la descarga a Colab utilizando gdown.

Base de datos vectorial:

Para generar la base de datos vectorial extraje el texto de los archivos pdf utilizando PdfReader de PyPDF2 y luego con re (Regular Expression) hice la limpieza de urls, caracteres no alfanuméricos, etc.

Hago el Split del texto con un tamaño de 1000 caracteres y una superposición de 200. Según las pruebas que hice fue el que mejor me armaba los textos teniendo en cuenta las explicaciones y los artículos de los distintos reglamentos.

Utilizo el modelo de embeddings de HuggingFace (sentence-transformers/paraphrase-multilingual-mpnet-base-v2) para generarlos y almacenar en la base de datos vectorial ChromaDB.

Al momento de crear la colección para la BD tenía problemas al ejecutar las distintas pruebas ya que al hacer alguna consecutiva me daba error de que ya existía y si ponía el delete me daba error la primera vez que lo ejecutaba porque no existía. La solución lógica fue verificar si existía y entonces hacer el delete.

Base de datos tabular:

Generé una base SQLite, genero la tabla directamente por código fuente en el propio Colab y luego ejecuto las instrucciones del archivo .sql para el llenado con los datos.

Luego hago una consulta y muestro todos los datos de la tabla para verificar que estén correctamente subidos.

Objeto para generar respuestas de lenguaje natural

Establezco y verifico la clave de API de OpenAI, que utilizo para inicializar un objeto ChatOpenAI para generar las respuestas de lenguaje natural basadas en los modelos de OpenAI. Elegí utilizar esta por sobre HuggingFace porque ofrece alta calidad y coherencia en generación de texto, puedo manejar contexto largo, tengo crédito en la aplicación y en este caso el uso fue muy económico ya que para todas las pruebas me consumió solamente 6 centavos de dólar.

Funciones para las distintas búsquedas según el tema:

Para la búsqueda en la base de datos SQLite generé el prompt con el contexto donde le digo que es experto en SQLite y le explico el diseño de la tabla en la base de datos para que sepa generar la consulta y me devuelva la respuesta. Luego se encarga de generar la consulta y devuelve el resultado como parte del contexto.

Para la búsqueda en la base de grafos, utilizo la librería SPARQLWrapper y configuro el punto de acceso SPARQL de Wikidata.
Genero funciones para que, según el wikidata_id, se ejecute la consulta SPARQL para obtener el nombre (label) en una y la descripción (description) de la entidad correspondiente en la otra en Wikidata.
Genero una tercera función (que llama a las otras dos) donde defino la plantilla del prompt para generar las consultas SPARQL, preparo los mensajes que se enviarán al modelo de chat ejecuto la consulta y proceso y devuelvo los resultados.

Para la búsqueda en la base vectorial generé una función a la que le paso la query y el modelo de embeddings (el de HuggingFace) genera la consulta y devuelve el resultado más cercano. Quedó un for que no quise cambiar porque si bien en este caso tomo el más cercano, podría tomar dos o tres y dar más de una respuesta. De todos modos el for funciona también con una sola respuesta. En este for se genera el contexto de resultado que se devuelve.

Clasificadores:

Basado en un modelo entrenado con ejemplos y embeddings:

Entrené un modelo con sklearn pasándole distintas preguntas clasificadas para los 3 tipos pero realmente no me funcionaba bien agregando varias preguntas y tratando de ser específico y claro para que el modelo comprenda pero entiendo que al ser tantas preguntas sobre un tema similar (Básquetbol) el modelo no llega a notar si es reglamento, estadística de mundiales o ninguno de los dos pero sí de basquet.

Basado en LLM:

Para este caso utilicé el objeto de ChatOpenAI también y me clasifica todas las preguntas que le hago de manera exacta. No tuve dudas de que esta es la opción óptima.

Generador de Respuesta

Aquí utilicé una función para generar la respuesta definitiva que mostrará el chatbot. Tuve que hacer una pequeña diferenciación de acuerdo al tipo de pregunta ya que para el caso de Wikidata si le pedía información de "manera precisa" limitaba mucho las respuestas o me decía que no tenía información suficiente y por otro lado, en el caso de la búsqueda en SQLite si no le pedía "de manera precisa" me respondía con algunos errores.

Front de Chatbot

Utilizo gradio y su método ChatInterface para ingresar las preguntas y mostrar las respuestas. Doy algunas preguntas de ejemplo para probar y existe la posibilidad de ingresar texto libremente con más preguntas.

➢ Conclusiones:

El proyecto presentó un desafío importante de llevar a la práctica las distintas consultas a las distintas fuentes de datos. Se utilizó mucho de lo enseñado en teoría y como base el código fuente proporcionado como ejemplos, pero también requirió de investigación sobre algunos temas en particular.

Es muy importante comprender los conceptos fundamentales para entender y utilizar las herramientas existentes pero muchos de los conceptos teóricos se fortalecieron al hacer el trabajo práctico.

A mí me resultó muy útil y me generó ideas para procesar información en PDF, por ejemplo una herramienta que procese CVs y traiga los resultados que mejor se ajustan a determinadas búsquedas.

El uso de la herramienta de OpenAI me pareció que si bien tiene un costo no es tan caro y las ventajas y calidad del procesamiento es muy destacable.

➢ Enlaces a modelos y librerías utilizadas:

Import gdown: https://pypi.org/project/gdown/

import re: https://docs.python.org/es/3/library/re.html

from PyPDF2 import PdfReader: https://pypi.org/project/PyPDF2/

from langchain.text_splitter import CharacterTextSplitter: https://api.python.langchain.com/en/latest/character/langchain_text_splitters.character.CharacterTextSplitter.html

from langchain.text_splitter import RecursiveCharacterTextSplitter: https://api.python.langchain.com/en/latest/character/langchain_text_splitters.character.RecursiveCharacterTextSplitter.html

from langchain.embeddings.huggingface import HuggingFaceEmbeddings: https://api.python.langchain.com/en/latest/embeddings/langchain_community.embeddings.huggingface.HuggingFaceEmbeddings.html

from langchain_openai import ChatOpenAI: https://pypi.org/project/langchain-openai/

from langchain_core.messages import HumanMessage: https://api.python.langchain.com/en/latest/messages/langchain_core.messages.human.HumanMessage.html

from langchain_core.messages import SystemMessage: https://api.python.langchain.com/en/latest/messages/langchain_core.messages.system.SystemMessage.html

from langchain.prompts import PromptTemplate: https://api.python.langchain.com/en/latest/prompts/langchain_core.prompts.prompt.PromptTemplate.html

import chromadb: https://pypi.org/project/chromadb/

import sqlite3: https://docs.python.org/es/3.8/library/sqlite3.html

import os: https://docs.python.org/es/3.10/library/os.html

from SPARQLWrapper import SPARQLWrapper, JSON: https://sparqlwrapper.readthedocs.io/_/downloads/en/stable/pdf/

from sklearn.model_selection import train_test_split: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

from sklearn.linear_model import LogisticRegression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

from sklearn.metrics import accuracy_score: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

from sklearn.metrics import classification_report: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

from sentence_transformers import SentenceTransformer: https://sbert.net/

import gradio: https://pypi.org/project/gradio/

Ejercicio 2:

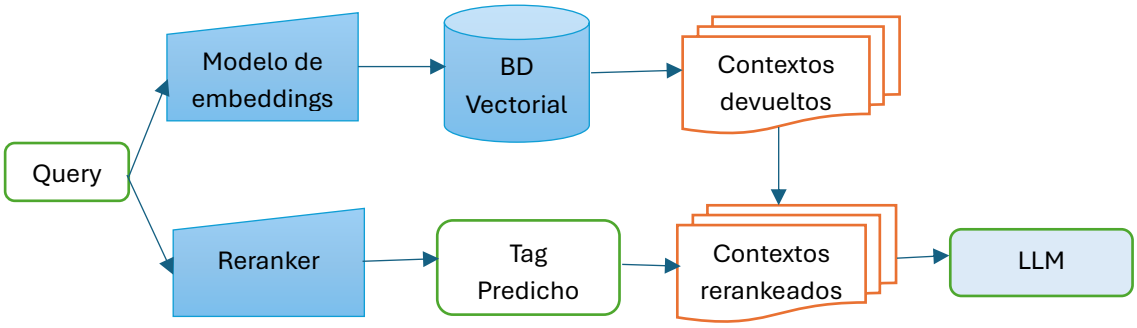
➢ Desarrollo de las preguntas:

1. Explique con sus palabras el concepto de Rerank y como impactaría en el desempeño de la aplicación. Incluir un diagrama de elaboración propia en la explicación.

   Reranking implica reevaluar y reorganizar los documentos o datos recuperados en función de su relevancia para la consulta. Este proceso perfecciona los resultados de la recuperación al priorizar aquellos documentos que son más apropiados en términos de contexto para la consulta. Esta selección mejorada mejora la calidad y precisión general de la información que el modelo utiliza para generar su salida final.

   La incorporación de rerankers mejora significativamente la precisión y relevancia de los resultados de búsqueda en tareas complejas de recuperación de información. Al aprovechar esta técnica, los modelos RAG pueden superar los desafíos relacionados con la pérdida de información y capturar de manera efectiva matices lingüísticos sutiles, lo que conduce a salidas más precisas.

   Considerando el caso particular del presente TP, el usuario consulta a un modelo de lenguaje para obtener información sobre el reglamento de básquetbol. A través del reranking, el sistema evalúa los documentos recuperados y los prioriza comparándolos con la consulta del usuario. Esto garantiza que la información proporcionada no solo sea precisa, sino también altamente relevante para las preocupaciones sobre la duda específica del usuario, mejorando así la confiabilidad y utilidad de la respuesta.



2. ¿En qué sección de su código lo aplicaría?
   Debería agregarlo luego de recuperar los resultados más relevantes (en mi trabajo traía solamente el más cercano y debería cambiarlo a que me traiga los n más cercanos).

Aquí debería inicializar un modelo de lenguaje con Cohere siguiendo la documentación especificada en https://docs.cohere.com/reference/rerank, configurar el reranker con CohereRerank y combinarlo con el recuperador base en un ContextualCompressionRetriever. Esta configuración comprime y rerankea los resultados de la recuperación, refinando la salida en función de la relevancia contextual.

➢ Detalle de las fuentes utilizadas y sus autores:

- Mejorando los sistemas avanzados de RAG utilizando reranking con LangChain: https://myscale.com/blog/es/maximizing-advanced-rag-models-langchain-reranking-techniques/
  Artículo publicado en el sitio de MyScale, motor de Base de Datos Vectorial para IA escalable.
- Mastering RAG: How to select a Reranking model: https://www.rungalileo.io/blog/mastering-rag-how-to-select-a-reranking-model
  Artículo publicado en el sitio de Galileo, una plataforma diseñada para ayudar a los equipos de inteligencia artificial y machine learning a garantizar la calidad y seguridad de sus modelos de recuperación y generación (RAG).