Trabajo Final Procesamiento del Lenguaje Natural

2023

Tomás Navarro Miñón

Link al Google Colab:
https://colab.research.google.com/drive/1iIbe1w8BnrBYp55Hm4YTWJqeeaFrYkMI?usp=sharing

Introducción:
La Generación Aumentada por Recuperación (RAG) es el concepto de proporcionar a los Modelos de Lenguaje de Gran Escala (LLMs) información adicional proveniente de una fuente de conocimiento externa. Esto les permite generar respuestas más precisas y contextuales, reduciendo las alucinaciones.

En términos simples, RAG es para LLMs lo que un examen con libro abierto es para los seres humanos. En un examen con libro abierto, se permite a los estudiantes llevar materiales de referencia, como libros de texto o apuntes, que pueden utilizar para buscar información relevante y responder a una pregunta. La idea detrás de un examen con libro abierto es que el examen se centra en las habilidades de razonamiento de los estudiantes en lugar de su capacidad para memorizar información específica.

Teniendo esto en cuenta nuestro sistema RAG lo hicimos en base a un autor de teoría política conocido como Nicolas Maquiavelo, él mismo fue autor de muchos libros que hoy en día se aplican en el campo de la política. Algunas de sus obras más conocidas son “El Príncipe”, “Discursos Sobre la Primera Década De Tito Livio”, entre otros.

¿Por qué realizar un sistema RAG de un autor de teoría política? Bueno tiene que ver con algo personal. Anteriormente yo estudiaba Relaciones Internacionales en la FCPOLIT. Algo que en lo personal no me gustaba de la carrera era que a la hora de rendir, tenía que leer absolutamente todo el autor, por más que muchas veces la mayoría de las cosas no eran importantes. Pero algo todavía peor es que si no entendía algo, a la única persona que le podía preguntar era al profesor en clases de consultas, por lo que si no podías ir,

seguramente no apruebes. Es por esto que viendo en google drive las cosas que tenía me incline a ayudar a los alumnos de aquella carrera que, como yo, no podrian ir a clases de consultas porque vivían lejos… De esta manera llego el Chatbot RAG: “Nicolás MaquiavRAGlo” (jaja malisimo el nombre)

Documentación TP NLP

1- Lo primero que hacemos es instalar las librerias necesarias para llevar a cabo el proyecto:

```
llama_index sentence-transformers pypdf langchain python-decouple
PyMuPDF gdown chromadb fpdf SPARQLWrapper
```

Luego el paso siguiente será la importación de todas las librerías a utilizar:

```
from langchain.embeddings.huggingface import HuggingFaceEmbeddings
from llama_index.embeddings import LangchainEmbedding
from llama_index import ServiceContext
from llama_index import VectorStoreIndex, SimpleDirectoryReader
from jinja2 import Template
from decouple import config
import chromadb
import os
from llama_index.node_parser.text import SentenceSplitter
from llama_index.schema import TextNode
from llama_index.vector_stores import ChromaVectorStore
from llama_index.storage.storage_context import StorageContext
import requests
from bs4 import BeautifulSoup
from fpdf import FPDF
import gdown
import shutil
```

2- Extracción de datos para la creación del sistema RAG
Link a los archivos del Google Drive:
https://drive.google.com/drive/folders/1AEDfhcL9aLA5vLIlLmrzeYl6zaL_yHur?usp=sharing
En este punto lo que hacemos es importar los PDFs que nos serviran para un posterior scrapping de los mismos y poder entrenar a nuestro chatbot especialista en Nicolas Maquiavelo.
Tambien en este punto realizamos web scraping con beautifulsoup a la pagina oficial de wikipedia para obtener mas informacion:
https://es.wikipedia.org/wiki/Nicol%C3%A1s_Maquiavelo
A su vez también obtenemos información realizando una consulta a una base de datos online SQL.

3- Obtenemos el modelo de LLM que vamos a utilizar desde hugging face en este caso LLama Index.
Y tambien cargamos el modelo de LangChain de Embeddings:
`sentence-transformers/paraphrase-multilingual-mpnet-base-v2`

```
def zephyr_instruct_template(messages, add_generation_prompt=True):
    # Definir la plantilla Jinja
    template_str = "{% for message in messages %}"
    template_str += "{% if message['role'] == 'user' %}"
    template_str += "<|user|>{{ message['content'] }}</s>\n"
    template_str += "{% elif message['role'] == 'assistant' %}"
    template_str += "<|assistant|>{{ message['content'] }}</s>\n"
    template_str += "{% elif message['role'] == 'system' %}"
    template_str += "<|system|>{{ message['content'] }}</s>\n"
    template_str += "{% else %}"
    template_str += "<|unknown|>{{ message['content'] }}</s>\n"
    template_str += "{% endif %}"
    template_str += "{% endfor %}"
    template_str += "{% if add_generation_prompt %}"
    template_str += "<|assistant|>\n"
    template_str += "{% endif %}"

    # Crear un objeto de plantilla con la cadena de plantilla
    template = Template(template_str)

    # Renderizar la plantilla con los mensajes proporcionados
    return template.render(messages=messages,
add_generation_prompt=add_generation_prompt)


# Aquí hacemos la llamada el modelo
def generate_answer(prompt: str, max_new_tokens: int = 768) -> None:
    try:
        # Tu clave API de Hugging Face
        api_key = config('HUGGINGFACE_TOKEN')

        # URL de la API de Hugging Face para la generación de texto
        api_url =
"https://api-inference.huggingface.co/models/HuggingFaceH4/zephyr-7b-be
ta"

        # Cabeceras para la solicitud
        headers = {"Authorization": f"Bearer {api_key}"}

        # Datos para enviar en la solicitud POST
        # Sobre los parámetros:
https://huggingface.co/docs/transformers/main_classes/text_generation
```

```
        data = {
            "inputs": prompt,
            "parameters": {
                "max_new_tokens": max_new_tokens,
                "temperature": 0.7,
                "top_k": 50,
                "top_p": 0.95
            }
        }

        # Realizamos la solicitud POST
        response = requests.post(api_url, headers=headers, json=data)

        # Extraer respuesta
        respuesta = response.json()[0]["generated_text"][len(prompt):]
        return respuesta

    except Exception as e:
        print(f"An error occurred: {e}")

# Esta función prepara el prompt en estilo QA
def prepare_prompt(query_str: str, nodes: list):
  TEXT_QA_PROMPT_TMPL = (
      "La información de contexto es la siguiente:\n"
      "---------------------\n"
      "{context_str}\n"
      "---------------------\n"
      "Dada la información de contexto anterior, y sin utilizar
conocimiento previo, responde la siguiente pregunta.\n"
      "Pregunta: {query_str}\n"
      "Respuesta: "
  )

  # Construimos el contexto de la pregunta
  context_str = ''
  for node in nodes:
      page_label = node.metadata["page_label"]
      file_path = node.metadata["file_path"]
      context_str += f"\npage_label: {page_label}\n"
      context_str += f"file_path: {file_path}\n\n"
      context_str += f"{node.text}\n"

  messages = [
```

```
      {
          "role": "system",
          "content": "Eres un asistente útil que siempre responde con
respuestas veraces, útiles y basadas en hechos.",
      },
      {"role": "user", "content":
TEXT_QA_PROMPT_TMPL.format(context_str=context_str,
query_str=query_str)},
  ]

  final_prompt = zephyr_instruct_template(messages)
  return final_prompt
```

Con el codigo anterior configuramos la llamada al modelo de LLM (llama index) para poder realizar prompts y obtener una respuesta. Tambien lo configuramos para que tome un rol, en este caso que sea un asistente.

4- Creamos la base de datos vectorial para almacenar los respectivos nodos creados que van a ayudar a encontrar información y etiquetar el contexto para nuestro sistema RAG

```
chroma_client = chromadb.EphemeralClient()
chroma_collection = chroma_client.create_collection("quickstart")
```

Creamos la colección de ChromaDB

```
text_parser = SentenceSplitter(
    chunk_size=1024,
)

text_chunks = []
doc_idxs = []
for doc_idx, doc in enumerate(documents):
    cur_text_chunks = text_parser.split_text(doc.text)
    text_chunks.extend(cur_text_chunks)
    doc_idxs.extend([doc_idx] * len(cur_text_chunks))

nodes = []
for idx, text_chunk in enumerate(text_chunks):
    node = TextNode(
        text=text_chunk,
    )
    src_doc = documents[doc_idxs[idx]]
    node.metadata = src_doc.metadata
    nodes.append(node)

for node in nodes:
    node_embedding = embed_model.get_text_embedding(
```

```
        node.get_content(metadata_mode="all")
    )
    node.embedding = node_embedding
```

En las líneas de código vistas anteriormente, a la data extraída de nuestros textos le realizamos Text Split, Tokenización, la transformamos en Nodos para poder añadirla a nuestra base de datos vectorial y realizamos su transformación a embeddings.

```
vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
storage_context =
StorageContext.from_defaults(vector_store=vector_store)
service_context =
ServiceContext.from_defaults(embed_model=embed_model,llm=None)
index = VectorStoreIndex(
    nodes, storage_context=storage_context,
service_context=service_context
)
```

En esta celda lo que realizamos es añadir todos los Nodos con sus respectivos embeddings en la base de datos vectorial

```
retriever = index.as_retriever(similarity_top_k=2)
```

Creamos el retriever que va ayudar a crear el modelo RAG (justamente esto lo que va a ser es encontrar la similaridad del coseno que más se aproxime a nuestra consulta transformada a embedding. Entonces el sistema con esto va a encontrar la respuesta y a su vez contextualizar la pregunta y la respuesta.

5- Ponemos en funcionamiento nuestro sistema RAG

```
print('Realizando llamada a HuggingFace para generar respuestas...\n')
while True:
  queries = input("Ingrese su pregunta para el Chat especialista en
Nicolas Maquiavelo: (si desea salir ingrese 'salir')")
  if queries=="salir":
    break
  nodes = retriever.retrieve(queries)
  final_prompt = prepare_prompt(queries, nodes)
  print('Pregunta:', queries)
  node_metadata = node.metadata
  file_name = node_metadata['file_name']
  # Ahora puedes imprimir el nombre del archivo
```

```
  print(f"La respuesta se puede encontrar en el archivo: {file_name}")
  print('Respuesta:')
  print(generate_answer(final_prompt))
  print('-----------------------------------------------------')
```

Para finalizar realizamos un bucle while donde nuestro chatbot completamente configurado va a responder todas las preguntas que le hagamos como si fuese un gran asistente virtual especialista en Nicolas Maquiavelo!