TUIA

PROCESAMIENTO DE LENGUAJE NATURAL

TRABAJO PRÁCTICO FINAL

2024

Alumno: PISTELLI, PABLO

Consignas:

En la primera parte del trabajo se creó un chatbot especializado en un tema a elección utilizando la técnica RAG (Retrieval Augmented Generation).

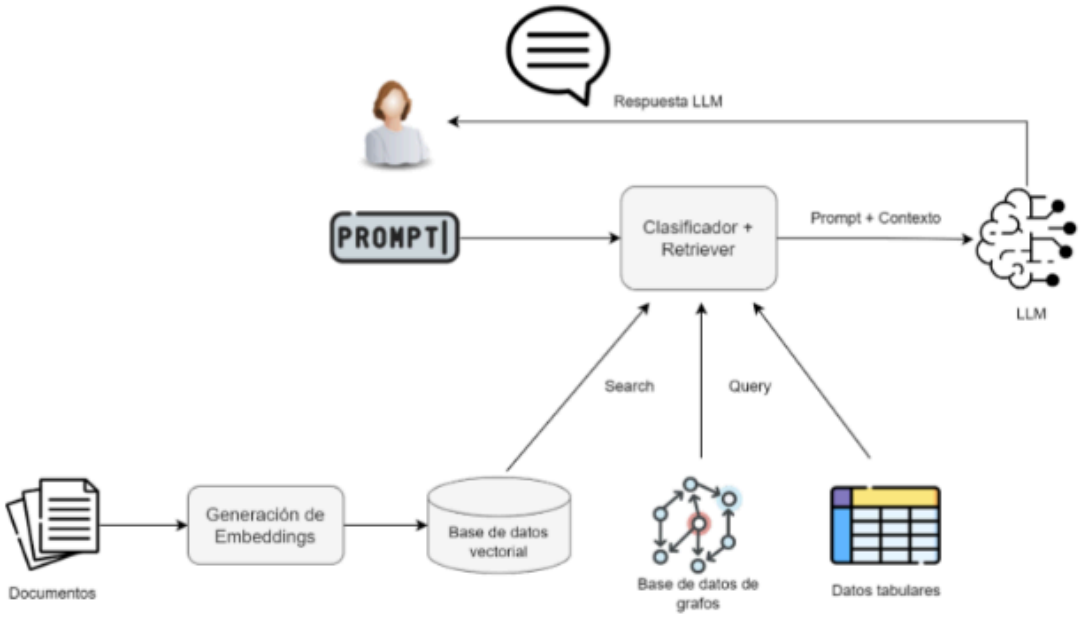
El objetivo es que el chatbot responda preguntas acerca de grupos musicales, su trayectoria, formaciones, estilos, grabaciones, etc. Como grupo de prueba elegí Foo Fighters, ya que por gustos personales, es de la cual tengo más información.

Como fuentes de conocimiento se utilizarán::

- Documentos de texto (PDF)
- Datos tabulares
- Base de datos de grafos

El usuario podrá interactuar con el chatbot en español y éste intentará responder a las consultas buscando información en las distintas fuentes provistas. El asistente podrá clasificar las consultas para elegir la fuente de información.

La ruta de nuestro chatbot será la siguiente:



En la segunda parte, se presenta una investigación sobre el estado del arte de las aplicaciones de agentes inteligentes, usando modelos LLM libres y se plantea la resolución de la problemática elegida con sistemas multi-agente.

Primera parte: Chatbot

Documentos de texto

Los archivos que se utilizarán son extractos de wikipedia, artículos de revistas y el libro The Storyteller de Dave Grohl.

Para la carga de los documentos necesarios para el proyecto, se accede a un link a Google Drive que contiene una carpeta con todos los archivos a utilizar

A partir de ese link, se descarga la carpeta y se mueven todos los archivos a una carpeta destino dentro del entorno colab.

Para la extracción del texto se utiliza una función llamada 'extraer_texto_pdf' usando la librería *pdfplumber* y se aplica a cada archivo PDF de la carpeta.

```
# Extracción de texto de PDF

import pdfplumber

def extraer_texto_pdf(ruta_pdf):
  with pdfplumber.open(ruta_pdf) as pdf:
    texto = ''
    for page in pdf.pages:
      texto += page.extract_text()
    return texto
```

Una vez extraído el texto de los documentos se acondiciona eliminando caracteres especiales, tildes y pasando los caracteres a minúsculas.

```
# Texto en minúsculas

texto_extraido = texto_extraido.lower()


# Elimino tildes

from unidecode import unidecode

texto_extraido = unidecode(texto_extraido)


# Elimino caracteres especiales

import re

texto_extraido = re.sub(r'[^a-zA-Z0-9\s]', '', texto_extraido)
```

```
# Elimino los saltos de línea (\n)
texto_limpio = texto_extraido.replace("\n", " ")
```

Luego se realiza la división de textos “en porciones” para su posterior procesamiento. Se realiza una segmentación recursiva con la librería LangChain.

Se probaron distintas longitudes de segmentación para analizar cambios en el desempeño del modelo.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter


text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=500,
    chunk_overlap=100,
    length_function=len,
    is_separator_regex=False,
)
splitted_text = text_splitter.split_text(texto_limpio)
```

Finalmente se generan los embeddings para el modelo BERT desarrollado por Google.

```
from transformers import BertModel, BertTokenizer
import torch
# Cargar el modelo BERT pre-entrenado y el tokenizador
modelo_bert =
BertModel.from_pretrained('bert-base-multilingual-cased')
tokenizer =
BertTokenizer.from_pretrained('bert-base-multilingual-cased')

def obtener_embeddings(text):
    embeddings = []
    for fragmento in text:
```

```
            tokens = tokenizer(fragmento, truncation=True,
            padding=True, return_tensors='pt', max_length=512)

            outputs = modelo_bert(**tokens)
            embedding_vector =
            outputs.last_hidden_state.mean(dim=1).squeeze()
            embeddings.append(embedding_vector.tolist())
    return embeddings
# Generación de embeddings (20 min aprox)
embeddings = obtener_embeddings(splitted_text)
```

Para realizar el almacenamiento de los embeddings se utilizará una base de datos vectorial mediante ChromaDB. Se crea una colección donde se almacenan los embeddings, los documentos y los ids.

```
# Generamos el cliente
import chromadb
chroma_client = chromadb.Client()
# Creamos la colección
collection = chroma_client.create_collection(name="cine_db")
if collection is not None:
    print("Collection already exists.")
else:
    collection = chroma_client.create_collection(name="db_foo")
# Generación de ids para almacenar en la chromaDB
ids = [f'id{i+1}' for i in range(len(splitted_text))]
len(ids)
# Se agregan los datos a la chromaDB
collection.add(
    embeddings=embeddings,
    documents=splitted_text,
    ids=ids)
```

Datos tabulares

Como datos tabulares se utilizaron estadísticas de reproducciones de Spotify y Youtube.

```
# Filtra solo los archivos csv
archivos_csv = [archivo for archivo in archivos_en_carpeta if archivo.endswith('.csv')]
carpeta = 'ff_docs'
ff_df = {}


# Itera sobre los archivos csv
for archivo_csv in archivos_csv:
    ruta_completa = os.path.join(carpeta, archivo_csv)
    df = pd.read_csv(ruta_completa, sep=';')
    ff_df[archivo_csv] = df
```

Se generan dos funciones que en su funcionamiento son iguales, pero que consultan diferentes tablas y retornan diferentes datos:

- top_songs: consulta estadísticas de Spotify y retorna título de la canción, álbum y cantidad de reproducciones.

```
def top_songs(df):
 '''
 Función que devuelve top X de canciones en Spotify
 '''
 # Seleccionar el top
 top = int(input("Ingrese el número de canciones del top: "))
 print('')
 print(f'Top {top} de canciones de Foo Fighters en Spotify:')
 print('')



 # Ordenar el DataFrame por puntaje de manera descendente
```

```
 df_ordenado = df.sort_values(by='Streams', ascending=False)


 top_df = df_ordenado.head(top)


 resultado = ""
 for index, row in top_df.iterrows():
     resultado += f"Título: {row['Song Title']}\n"
     resultado += f"Album: {row['Album']}\n"
     resultado += f"Reproducciones: {row['Streams']}\n"


     resultado += "\n"


 return resultado
```

- top_videos: consulta estadísticas de Youtube y retorna título del video, fecha de publicación y cantidad de reproducciones.

```
def top_videos(df):
 '''
 Función que devuelve top X de videos en Youtube
 '''
 # Seleccionar el top
 top = int(input("Ingrese el número de videos del top: "))
 print('')
 print(f'Top {top} de videos de Foo Fighters en Youtube:')
 print('')



 # Ordenar el DataFrame por puntaje de manera descendente
 df_ordenado = df.sort_values(by='Views', ascending=False)


 top_df = df_ordenado.head(top)
```

```
 resultado = ""
 for index, row in top_df.iterrows():
     resultado += f"Título: {row['Video']}\n"
     resultado += f"Publicación: {row['Published']}\n"
     resultado += f"Reproducciones: {row['Views']}\n"


     resultado += "\n"


 return resultado
```

Base de datos de Grafos

Para la base de datos de Grafos se utilizará Wikidata. Es una base de datos gratuita, colaborativa y multilingüe, que recopila datos estructurados para brindar soporte a proyectos de Wikimedia, como Wikipedia. Se realizan las consultas utilizando la librería SPARQLWrapper.

Las consultas fueron diseñadas siguiendo la ayuda de WikiData, utilizando el buscador para obtener los números de parámetros y probadas en https://query.wikidata.org/.

```
def execute_sparql_query(query):
   sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
   sparql.setQuery(query)
   sparql.setReturnFormat(JSON)
   results = sparql.query().convert()
   return results['results']['bindings']


# Consulta SPARQL
sparql_query = """
   SELECT ?album ?albumLabel ?releaseDate ?genreLabel ?artistLabel
?duration ?coverArt ?membersLabel ?producerLabel ?recordLabelLabel
?numberOfTracks
   WHERE {
     ?album wdt:P31 wd:Q482994.
     ?album wdt:P175 wd:Q483718. # Foo Fighters
```

```
      ?album wdt:P577 ?releaseDate.
      OPTIONAL { ?album wdt:P136 ?genre. }
      OPTIONAL { ?album wdt:P175 ?artist. }
      OPTIONAL { ?album wdt:P2047 ?duration. }
      OPTIONAL { ?album wdt:P18 ?coverArt. }
      OPTIONAL { ?album wdt:P527 ?members. }
      OPTIONAL { ?album wdt:P162 ?producer. }
      OPTIONAL { ?album wdt:P264 ?recordLabel. }
      OPTIONAL { ?album wdt:P1108 ?numberOfTracks. }
      SERVICE wikibase:label { bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }
   }
   ORDER BY DESC(?releaseDate)
"""


# Ejecutar la consulta SPARQL
discos = execute_sparql_query(sparql_query)
```

Esta función devuelve información de los discos y es procesada por otra que extrae la información de un álbum en particular.

```
def buscar_disco(datos, disco):
   resultados = []
   for dato in datos:
       if dato['albumLabel']['value'].lower() == disco.lower():
           resultados.append(dato)


   # Procesar los resultados
   for result in resultados:
     print("Álbum:", result['albumLabel']['value'])
     print("Fecha de lanzamiento:", result['releaseDate']['value'])
     if 'genreLabel' in result:
         print("Género:", result['genreLabel']['value'])
```

```
        if 'artistLabel' in result:
            print("Artista:", result['artistLabel']['value'])
        if 'duration' in result:
            duracion_en_minutos = int(result['duration']['value']) // 60
            print("Duración:", duracion_en_minutos)
        if 'coverArt' in result:
            print("Portada del álbum:", result['coverArt']['value'])
        if 'membersLabel' in result:
            print("Miembros de la banda:",
result['membersLabel']['value'])
        if 'producerLabel' in result:
            print("Productor:", result['producerLabel']['value'])
        if 'recordLabelLabel' in result:
            print("Sello discográfico:",
result['recordLabelLabel']['value'])
        if 'numberOfTracks' in result:
            print("Número de canciones:",
result['numberOfTracks']['value'])
        print()


    return resultados
```

Clasificador y Retriever

Se utilizarán plantillas Jinja para guardar información de formato.

Para el modelo de generación de texto como chat se usará ***"HuggingFaceH4/zephyr-7b-beta"***, a través de su API en Hugging Face.

Este modelo está entrenado para actuar como asistente útil.

Se utiliza una función para definir los templates Jinja.

```
def zephyr_chat_template(messages, add_generation_prompt=True):
```

Y se utiliza otra función para llamar al modelo de HuggingFace y elegir la fuente que se considere correcta para cada pregunta.

```
def generar_respuesta(prompt: str, api_key, max_new_tokens: int = 768) -> None:
    '''
    Función que llama al Modelo de HuggingFace y elige la fuente que considera correcta para cada pregunta.
    Parámetros: prompt  --> la pregunta que queremos que el modelo responda
                api_key --> nuestra clave API de HuggingFace
    '''
```

Para el diseño y formato de prompts, se utilizaron diferentes few-shots con posibles preguntas que recibirá el agente.

```
                                    {
                                      "role": "system",
                                      "content": "Soy un asistente especializado en Foo Fighters, capaz de identificar álbums y proporcionar información sobre ellos."
                                    },
                                    {
                                      "role": "user",
                                      "content": "¿De qué año es el album One by One?"
                                    },
                                    {
                                      "role": "assistant",
                                      "content": "Album: [One by One]"
                                    },
                                     {
                                      "role": "user",
                                      "content": "Brindame información sobre el ultimo album."
```

```
                    },
                    {
                      "role": "assistant",
                      "content": "Album: [But Here We
Are]"
                    },
                    {
                      "role": "user",
                      "content": "¿Quien es el productor
de Medicine by Midnight?"
                    },
                    {
                      "role": "assistant",
                      "content": "Album: [Medicine by
Midnight]"
                    },
                    {
                      "role": "user",
                      "content": "Quiero saber cuáles son
las 20 canciones más reproducidas en Spotify de Foo Fighters"
                    },
                    {
                      "role": "assistant",
                      "content": "Ranking canciones"
                    },
                    {
                      "role": "user",
                      "content": "¿Cuales son los videos
más reproducidos de Foo Fighters en Youtube?"
                    },
                    {
                      "role": "assistant",
                      "content": "Ranking videos"
```

```
                                    },
                                    {
                                      "role": "user",
                                      "content": "¿En qué año se formó Foo
Fighters?"
                                    },
                                    {
                                      "role": "assistant",
                                      "content": "Historia"
                                    },
                                    {
                                      "role": "user",
                                      "content": "¿Qué bandas fueron
influencia para Dave Grohl?"
                                    },
                                    {
                                      "role": "assistant",
                                      "content": "Historia"
                                    },
                                    {
                                      "role": "user",
                                      "content": "¿Donde nació Dave
Grohl?"
                                    },
                                    {
                                      "role": "assistant",
                                      "content": "Historia"
                                    },
```

Contexto y clasificación

Para saber qué fuentes de datos utilizar como contexto para generar una respuesta, se va a pasar la información obtenida del clasificador a cada fuente, según corresponda.

Primero, se cambia el formato de la respuesta para que solamente nos devuelva lo que nos interesa y acceder de una forma más precisa al contexto utilizando la funcion formatear_respuesta.

Luego, con la función devolver_contexto se obtiene el contexto de la consulta, ingresando como parámetros la respuesta limpia y el prompt.

A continuación se prepara el prompt en estilo QA y se agrega el contexto y la fuente de información a la que accede el asistente para responder la pregunta.

El resultado es el siguiente:

```
# Ejemplo de aplicación:
respuesta_final = preparar_prompt(prompt3, contexto3, api_key=api_key)


print(f'Prompt: {prompt3}\n')
print('-------------------------------------------------------------------
-------')
print(f'Contexto: {contexto3}')
print(fuente3)
print('-------------------------------------------------------------------
-------')
# print(f'Respuesta:\n {respuesta_final}')
print(respuesta_final)
```

```
Prompt: ¿Cuál es el video más reproducido en Youtube?


--------------------------------------------------------------------------
Contexto: Título: Foo Fighters - The Pretender
Publicación: 2009/10
Reproducciones: 568120968



Fuente: Datos tabulares
--------------------------------------------------------------------------
```

```
La canción "The Pretender" de la banda Foo Fighters, publicada en el
año 2009 y que cuenta con más de 568 millones de reproducciones en
Youtube, es el video
```

Chatbot

Para simplificar el acceso al asistente, se unen todas las funciones anteriormente instanciadas, en una sola función:

```
def chatbot(prompt, api_key):
    '''
    Función para acceder al funcionamiento del chatbot
    Llama a las funciones:
            generar_respuesta()
            formatear_respuesta()
            devolver_contexto()
            preparar_prompt()


    Parámetros: prompt --> la pregunta que queremos que el chatbot
nos responda
    '''
```

Ejemplo de funcionamiento del chatbot

```
prompt = '¿Cuáles son las primeras 5 canciones del ranking de Spotify?'
print(f'Pregunta:\n{prompt}')
print('----------------------------------------------------------------
--------')
respuesta, fuente, contexto = chatbot(prompt, api_key)


# Imprimimos el resultado
print('----------------------------------------------------------------
--------')
```

```
print(fuente)
if fuente != 'Error al identificar la fuente':

print('--------------------------------------------------------------------
--------')
 print(f'Contexto:\n{contexto}')
print('--------------------------------------------------------------------
--------')
print(f'Respuesta:\n{respuesta}')
```

```
Pregunta:

¿Cuáles son las primeras 5 canciones del ranking de Spotify?

--------------------------------------------------------------------------

Ranking canciones

Ingrese el número de canciones del top: 5


Top 5 de canciones de Foo Fighters en Spotify:


--------------------------------------------------------------------------

Fuente: Datos tabulares

--------------------------------------------------------------------------

Contexto:

Título: Everlong

Album: The colour and the shape

Reproducciones: 1078134925


Título: The Pretender

Album: Echoes, silence, patience & grace

Reproducciones: 737175107


Título: Best of You
```

Album: In your honor

Reproducciones: 605359690

Título: Learn to Fly

Album: There is nothing left to lose

Reproducciones: 548061646

Título: My Hero

Album: The colour and the shape

Reproducciones: 388855963

Respuesta:

La canción con más reproducciones entre las mencionadas es "Everlong", perteneciente al álbum "The colour and the shape". Luego vienen "The Pretender" de "Echoes, silence, patience & grace", "Best of You" de "In your honor", "Learn to Fly" de "There is nothing left to lose" y por último "My Hero" también de "The colour and the shape". Así que, en resumen, las primeras 5 canciones del ranking de Spotify son:

1. "Everlong" - Foo Fighters
2. "The Pretender" - Foo Fighters
3. "Best of You" - Foo Fighters
4. "Learn to Fly" - Foo Fighters
5. "My Hero" - Foo Fighters

CONCLUSIÓN

En resumen, el trabajo práctico concluye con la implementación de un asistente virtual especializado en cine, respaldado por una infraestructura de bases de datos que incluye sistemas vectoriales, de grafos y tabulares.
La elección de estas bases de datos ha permitido una gestión eficiente de la información y un desempeño optimizado del asistente.

Segunda parte: Agentes

Un agente inteligente es un sistema perceptivo que puede interpretar y procesar la información que recibe de su entorno y actuar en consecuencia de acuerdo a los datos que obtiene y procesa. Su forma de actuación es lógica y racional basándose en el comportamiento normal de un sistema concreto. Utiliza sensores para recibir la información y actuadores para ejercer sus funciones.

Un sistema multiagente es un sistema compuesto por múltiples agentes inteligentes que interactúan entre ellos. Los sistemas multiagente pueden ser utilizados para resolver problemas que son difíciles o imposibles de resolver para un agente individual.

Los ámbitos en los que la investigación de sistemas multiagente puede ofrecer un enfoque adecuado incluyen, por ejemplo, el comercio en línea, la respuesta a desastres y el modelado de estructuras sociales.

Tipos de agentes:

- **Agente de reactivo simple**: Cuando una percepción coincide con una regla programada, el agente responde según la forma en la que fue programado. (Condición -> acción)
- **Agente reactivo basado en modelo**: Puede simular su acción de respuesta para estudiar su comportamiento y sus consecuencias en el espacio de actuación.
- **Agente basado en metas**: Combina los dos tipos de agentes previos. Tiene un objetivo concreto y busca la vía más óptima y planifica las acciones para cumplir su propósito.
- **Agente basado en utilidad**: Puede medir el valor de su comportamiento en el cumplimiento de las metas establecidas. Garantiza alta calidad en sus acciones.
- **Agente que aprende**: Aprende de sus acciones mientras está en funcionamiento. Tiene la capacidad de interactuar en entornos que no conoce.
- **Agente de consulta**: Responde consultas por parte de las personas que interactúan con este sistema. Crea varios agentes y divide la pregunta del usuario para dar una

respuesta adecuada. En caso de que los agentes no puedan responder la pregunta, crea más agentes y busca en más bases de datos para dar una solución.

Estado del arte de agentes inteligentes usando modelos LLM libres:

Los modelos de lenguaje basados en aprendizaje profundo, como GPT, Perplexity y PaLM, son una de las tecnologías más avanzadas en el procesamiento del lenguaje natural (PLN) y se utilizan en una variedad de aplicaciones de agentes inteligentes. Son conocidos por su capacidad para generar texto similar al humano y realizar tareas de PLN, como traducción automática, generación de texto y respuesta a preguntas.

Sin embargo, es importante tener en cuenta que los modelos LLM (Large Language Models) libres, como los de OpenAI (GPT), requieren un alto costo computacional y de recursos de entrenamiento, lo que limita su disponibilidad para todos los desarrolladores. A pesar de esto, han surgido varias aplicaciones de agentes inteligentes que aprovechan estos modelos LLM libres en diversas áreas:

Asistentes Virtuales: Los modelos LLM libres se utilizan para crear asistentes virtuales avanzados que pueden entender consultas complejas, realizar tareas específicas y brindar respuestas contextualmente relevantes.

Chatbots: Los chatbots impulsados por modelos LLM libres son capaces de mantener conversaciones más naturales y significativas con los usuarios, ya que tienen una comprensión más profunda del lenguaje natural.

Generación de Texto: Los modelos LLM libres pueden generar texto de manera creativa y coherente en una variedad de estilos y temas. Se utilizan en aplicaciones de redacción automática, creación de contenido, resumen de documentos y más.

Análisis de Sentimientos y Opiniones: Estos modelos pueden analizar grandes volúmenes de texto para identificar tendencias, opiniones y emociones. Se utilizan en aplicaciones de análisis de sentimientos en redes sociales, revisión de productos, encuestas en línea y más.

Traducción Automática: también se utilizan en sistemas de traducción automática para mejorar la precisión y la fluidez de las traducciones entre diferentes idiomas.

Es importante destacar que el desarrollo y la implementación de aplicaciones de agentes inteligentes utilizando modelos LLM libres deben considerar aspectos éticos, como la privacidad de los datos, el sesgo en el lenguaje y la transparencia en el uso de la inteligencia artificial.

Fuentes:

- https://es.wikipedia.org/wiki/Sistema_multiagente
- https://www.b2chat.io/blog/b2chat/sistemas-multiagente-que-son-como-funcionan/
- https://weremote.net/retos-asistentes-virtuales/
- https://fastercapital.com/es/tema/%C2%BFc%C3%B3mo-puede-hacer-que-sus-asistentes-virtuales-sean-exitosos.html