TUIA NLP 2024

TRABAJO PRÁCTICO FINAL

ESTUDIANTE:

MIRIAN YAÑEZ

TUIA NLP 2024

TRABAJO PRÁCTICO FINAL

Ejercicio 1:

Introducción

Este trabajo práctico tiene como objetivo desarrollar un chatbot experto en fútbol utilizando la técnica RAG (Retrieval Augmented Generation). La técnica de RAG combina la recuperación de información y la generación de texto para crear un chatbot capaz de responder preguntas basadas en diversas fuentes de datos.

La implementación se ha llevado a cabo en un entorno Google Colab y utiliza diversas fuentes de conocimiento, tales como:

- **Documentos PDF**: Información relacionada con el reglamento del fútbol.
- **Archivo CSV**: Resultados de partidos de fútbol.
- **Base de datos de grafos**: Información sobre jugadores obtenida de Wikidata.

El sistema permite a los usuarios hacer preguntas en español o inglés, y el chatbot responde utilizando la fuente de datos más relevante para proporcionar respuestas precisas y contextualmente adecuadas.

Ejecución del Programa

Para ejecutar el programa correctamente, sigue estos pasos detallados:

1. **Descargar el archivo desde GitHub**:
    - Ingresa a la siguiente URL:

      https://github.com/Mirian2550/NLP_tp/tree/main/TP2.
    - Selecciona el archivo correspondiente para descargar:

      tp2NLP_yañez.ipynb.
    - Haz clic en el botón de descarga (ícono de flecha hacia abajo) y espera a que la descarga se complete.
2. **Subir el archivo a Google Colab**:
    - Abre tu navegador y dirígete a Google Colab.

      https://colab.research.google.com/?hl=es
    - Haz clic en "Subir" para cargar tu propio archivo.

- Presiona el botón "Explorar" y selecciona el archivo tp2NLP_yañez.ipynb que descargaste anteriormente desde la carpeta de descargas de tu computadora.
- Espera a que el archivo se cargue completamente en Google Colab.

3. **Configurar el entorno de ejecución**:
   - En Google Colab, ve a la casilla “Entorno de ejecución” en el menú superior.
   - Selecciona la opción “Cambiar tipo de entorno de ejecución”.
   - En el cuadro de diálogo que aparece, selecciona “GPU” en el desplegable "Acelerador de hardware" y asegúrate de que la opción "T4 GPU" esté seleccionada.
   - Haz clic en "Guardar" para aplicar los cambios.
4. **Ejecutar todas las celdas**:
   - Nuevamente, dirígete a la casilla “Entorno de ejecución” en el menú superior.
   - Selecciona la opción “Ejecutar todas” para ejecutar todas las celdas del notebook. También puedes presionar Ctrl+F9 en tu teclado para ejecutar todas las celdas de manera rápida.

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

Configuración del entorno e instalación de librerías

Primero, configuré el entorno necesario para el proyecto. Instalé e importé las librerías necesarias.

Descarga de archivos PDF y CSV desde Google Drive

Para obtener los datos necesarios de manera dinámica, se descargan los archivos PDF y CSV desde Google Drive.

Uso de SPARQL y Wikidata para la Base de Datos de Grafos

Opté por obtener información desde Wikidata utilizando SPARQL. Procesé y formateé los resultados de las consultas SPARQL, extrayendo información relevante sobre jugadores de fútbol, como nombre, fecha de nacimiento, nacionalidad, equipo, posición, altura, peso, país del que posee ciudadanía y la edad calculada.

Detección del idioma y traducción

Para detectar el idioma del texto, utilicé la función detect de langdetect. La instancia del traductor GoogleTranslator con detección automática del idioma fuente y el idioma objetivo configurado a español. Si el idioma detectado es inglés, el texto se traduce al español. Si el texto ya está en español, se deja sin cambios. La función devuelve una tupla que incluye el idioma detectado y el texto traducido, si es necesario.

Construcción del clasificador basado en los conceptos de la unidad 3:

Con el objetivo de construir el mejor clasificador posible, realicé una combinación de técnicas entre lo aprendido en la unidad 3 y la construcción de un modelo de perceptrón multicapas con su correspondiente ajuste de hiperparámetros. A continuación voy a describir los pasos realizados para dicha construcción

Definición de estructuras de preguntas

Para poder clasificar, definí tres listas (estructuras_jugadores, estructuras_resultados, estructuras_reglamento) que contienen diferentes formas de preguntas para cada categoría.

Generación de preguntas aleatorias

Generé 1000 preguntas aleatorias con Faker, seleccionando aleatoriamente una categoría y una estructura de pregunta correspondiente.

Creación y guardado del DataFrame

Creé un DataFrame de pandas con dos columnas: 'Consulta' y 'Categoría', donde almacené las preguntas generadas y sus respectivas categorías. Luego, guardé el DataFrame en un archivo CSV llamado consultas.csv.

Preprocesamiento y generación de Embeddings

Descargué las palabras vacías (stop words) en español con NLTK. Cargué el modelo de transformación de oraciones all-mpnet-base-v2 de sentence-transformers. Utilicé las siguientes funciones:

- clean_text: Limpia el texto eliminando caracteres especiales, convirtiéndolo a minúsculas y eliminando palabras vacías.
- preprocess_texts: Aplica la función de limpieza a una lista de textos.
- get_bert_embeddings: Genera embeddings de BERT para una lista de textos.

Optimización y entrenamiento del clasificador

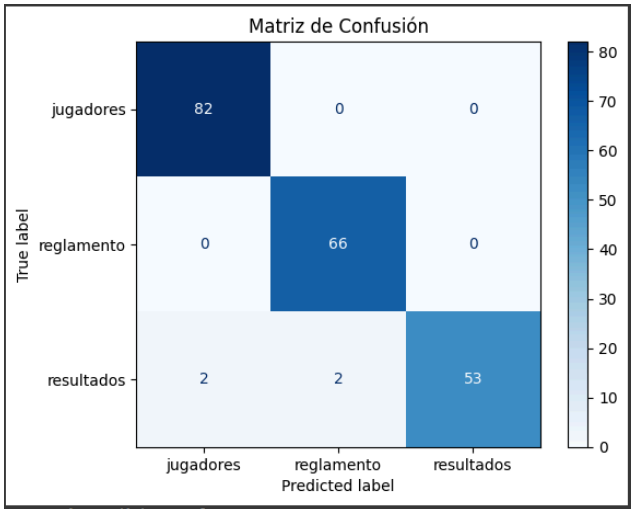
Definí los hiperparámetros a optimizar utilizando Optuna. Probé diferentes configuraciones de redes neuronales y evalué su precisión en el conjunto de datos de prueba. Entrené el modelo de clasificación, preprocesando los datos, generando embeddings, realizando oversampling para balancear las clases,

dividiendo los datos en conjuntos de entrenamiento y prueba, optimizando los hiperparámetros y entrenando el modelo final.

Evaluación del modelo

Utilicé la función mostrar_informacion para calcular y mostrar las métricas de evaluación del modelo tanto para el conjunto de entrenamiento como para el de prueba. También mostré la matriz de confusión.

```
Métricas de Entrenamiento
Precisión: 0.9755501222493888
F1-Score: 0.9753323398912443
Precisión (precision): 0.9764466496525107
Recall: 0.9755501222493888
Métricas de Prueba
Precisión: 0.9804878048780488
F1-Score: 0.9802646160642836
Precisión (precision): 0.9810070369611259
Recall: 0.9804878048780488
```



Las métricas de evaluación, con una precisión del 97%, indican que el modelo tiene un muy buen desempeño general. Sin embargo, la matriz de confusión revela que la categoría "jugadores" es la que mejor se ajusta, seguida por "reglamento", mientras que "resultados" presenta algunos errores de clasificación.

Construcción de clasificador utilizando lo aprendido en la unidad 6:

Para implementar la clasificación **zero-shot**, primero inicialicé el pipeline de zero-shot-classification utilizando el modelo preentrenado **facebook/bart-large-mnli** de la librería **transformers**. Este modelo es ideal porque permite clasificar texto en categorías especificadas por el usuario sin requerir un entrenamiento específico en esas categorías. Luego, definí las etiquetas de las categorías en las que quería clasificar las consultas, eligiendo "**resultados**", "**jugadores**" y "**reglamento**". Después, desarrollé una función llamada clasificar_consulta que recibe una consulta como entrada. Dentro de esta función, utilicé el clasificador para determinar a cuál de las categorías definidas

pertenece la consulta, basándome en la puntuación más alta. El resultado es la categoría más probable, que se devuelve como salida de la función. Este enfoque me permite clasificar eficientemente cualquier consulta en las categorías seleccionadas sin necesidad de un conjunto de datos de entrenamiento específico para estas categorías.

```python
from transformers import pipeline

# Inicializar el pipeline de zero-shot classification con un modelo preentrenado
classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")
# Definir las etiquetas de categorías
labels = ["resultados", "jugadores", "reglamento"]
# Función para clasificar consultas
def clasificar_consulta(consulta):
    result = classifier(consulta, candidate_labels=labels)
    # Obtener la etiqueta con la mayor puntuación
    return result['labels'][0]
```

Comparación de Clasificadores

Comparé el rendimiento de los dos clasificadores: el entrenado y el de zero-shot. Tras realizar las pruebas, los resultados mostraron que el clasificador entrenado es más preciso en sus predicciones que el clasificador zero-shot. Por esta razón, opté por utilizar el clasificador entrenado.

```
pregunta: ¿Quién es Messi?
entrenado:  jugadores  zero-shot:  resultados
pregunta: ¿qué es un penal?
entrenado:  reglamento  zero-shot:  resultados
pregunta: ¿qué es un partido de fútbol?
entrenado:  reglamento  zero-shot:  resultados
pregunta: ¿cuál es el resultado de river?
entrenado:  resultados  zero-shot:  resultados
pregunta: ¿qué es un partido de fútbol 11?
entrenado:  reglamento  zero-shot:  resultados
pregunta: ¿quién juega de delantero?
entrenado:  reglamento  zero-shot:  jugadores
pregunta: ¿cómo salio boca?
entrenado:  reglamento  zero-shot:  resultados
pregunta: ¿qué significa la tarjeta roja?
entrenado:  reglamento  zero-shot:  resultados
pregunta: ¿cuánto dura un partido?
entrenado:  resultados  zero-shot:  resultados
pregunta: ¿quién es batistuta?
entrenado:  resultados  zero-shot:  resultados
```

Fuente de datos: PDF

A continuación voy a explicar el proceso de construcción y almacenado de los embedding que posteriormente serán utilizados para generar el contexto cuando la pregunta corresponda a la categoría “reglamento”

Almacenamiento y búsqueda en ChromaDB

Generé embeddings y los almacené en ChromaDB. Verifiqué si el directorio de almacenamiento existía y lo eliminé para asegurar un entorno limpio.

Procesamiento de texto

Extraje texto de archivos PDF, lo normalicé eliminando acentos y convirtiéndolo a minúsculas, eliminé espacios en blanco adicionales y caracteres especiales, y dividí el texto en chunks.

Configuración y procesamiento de archivos PDF

Configuré ChromaDB y el modelo de embeddings. Definí los archivos PDF a procesar junto con sus rangos de páginas, y procesé cada archivo PDF, almacenando los embeddings generados en ChromaDB.

Fuente de datos: CSV

Reconocimiento de Entidades de Equipos de Fútbol

Para mejorar el reconocimiento de entidades de equipos de fútbol, implementé patrones de coincidencia tanto para los nombres oficiales como para los alias de cada club. Estos patrones se añadieron al EntityRuler de spaCy para que el modelo pudiera reconocerlos como entidades del tipo "TEAM". Dado que el uso de análisis POST o NER estándar no detectaba correctamente estas entidades debido a las limitaciones en el contexto de oraciones muy específicas y la ambigüedad del lenguaje, entrené spaCy para agregar una nueva entidad que representa a los clubes de fútbol. Esto permitió determinar con mayor precisión la presencia de equipos en una oración.

Generación de prompts y respuestas

Definí una lista de mensajes, generé un prompt utilizando zephyr_instruct_template, llamé a generate_answer para obtener una respuesta del modelo, e imprimí la respuesta generada. Interactué con un modelo de generación de texto a través de la API de Hugging Face.

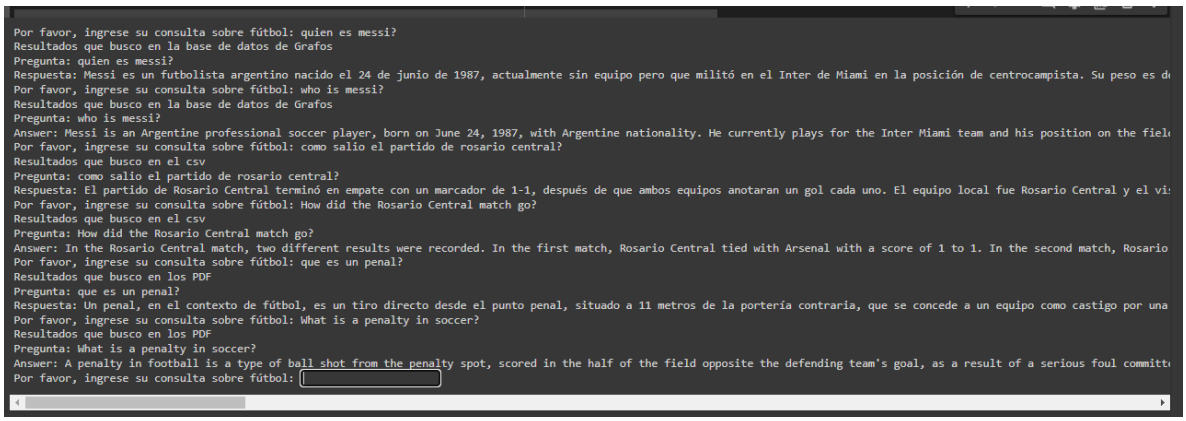
Preparación del Prompt

Utilicé prepare_prompt para generar un prompt formateado para pasarlo a un modelo de lenguaje para obtener una respuesta. Esta función facilita la creación de prompts bien estructurados que incluyen tanto el contexto relevante como la consulta del usuario. Al usar plantillas de Jinja, aseguré que los mensajes estuvieran correctamente formateados y listos para ser procesados por el modelo.

Sistema Interactivo Final

Implementé un sistema interactivo que toma una consulta del usuario, clasifica la consulta en una de las categorías (resultados, reglamento, jugadores), y obtiene información relevante desde varias fuentes (un archivo CSV, una base de datos de

embeddings de los PDF, o una base de datos de grafos). Finalmente, genera una respuesta usando un modelo de lenguaje y la presenta al usuario. Preparo el prompt final con la información de contexto, genero una respuesta usando la función generate_answer, traduzco la respuesta al inglés si la consulta original estaba en inglés, e imprimo la respuesta al usuario.



Conclusión

A lo largo de este proyecto, enfrenté varias dificultades técnicas que me obligaron a profundizar en diferentes áreas para lograr una solución efectiva. Uno de los primeros desafíos fue utilizar SPARQL y Wikidata como base de datos de grafos, lo cual implicó aprender el lenguaje de consulta SPARQL para extraer información relevante sobre jugadores de fútbol. La detección del idioma y su traducción también presentaron un reto, especialmente para frases muy cortas, donde el modelo tenía dificultades para diferenciar los idiomas correctamente.

Otra dificultad significativa fue la implementación de filtros sobre el CSV para asegurar la calidad de los datos. Descubrí que una mayor cantidad de información precisa resultaba en mejores métricas y un mejor desempeño del modelo. El proceso de construcción del clasificador también fue desafiante; entrenar el modelo y encontrar las mejores etiquetas para las consultas requirió múltiples iteraciones y ajustes. Opté por el modelo entrenado debido a su mayor precisión en comparación con el clasificador zero-shot, lo que me permitió alcanzar una precisión del 97%.

En conclusión, este proyecto me permitió adquirir conocimientos profundos en el manejo de bases de datos de grafos, técnicas de detección y traducción de idiomas, y en la optimización de clasificadores de texto. A pesar de los desafíos iniciales, logré desarrollar un sistema sencillo y eficiente capaz de responder consultas y proporcionar respuestas contextualizadas. Esta experiencia destacó la importancia de la calidad de los datos y la necesidad de una optimización continua para lograr un rendimiento óptimo.

Enlaces a la documentación oficial o repositorios de GitHub de los modelos y librerías utilizados

- **gdown**: para descargar archivos desde Google Drive.
  - https://pypi.org/project/gdown/
  - https://github.com/wkentaro/gdown
- **langchain**: Herramientas para el procesamiento y manejo de textos.
  - https://pypi.org/project/langchain/
  - https://github.com/langchain-ai/langchain
- **PyPDF2**: para manipulación y extracción de texto desde archivos PDF.
  - https://pypi.org/project/PyPDF2/
- **sentence-transformers**: Generación de embeddings a partir de oraciones.
  - https://pypi.org/project/sentence-transformers/
- **chromadb**: Base de datos para almacenar y recuperar embeddings.
  - https://pypi.org/project/chromadb/
- **SPARQLWrapper**: Ejecución de consultas SPARQL para interactuar con bases de datos de grafos.
  - https://github.com/RDFLib/sparqlwrapper
- **langdetect**: Detección del idioma de las consultas.
  - https://pypi.org/project/langdetect/
- **googletrans**: Traducción automática entre idiomas.
  - https://pypi.org/project/googletrans/
- **httpx**: Cliente HTTP para interactuar con APIs.
  - https://pypi.org/project/httpx/
- **faker**: Generación de datos falsos para pruebas.
  - https://pypi.org/project/Faker/
- **optuna**: Optimización de hiperparámetros.
  - https://pypi.org/project/optuna/
- **llama-index**: Herramientas para trabajar con índices y embeddings.
  - https://pypi.org/project/llama-index/

- **llama-index-embeddings-huggingface**: Herramientas para trabajar con embeddings de Hugging Face.
  - https://docs.llamaindex.ai/en/stable/examples/embeddings/huggingface/
- **python-decouple**: Configuración de variables de entorno.
  - https://pypi.org/project/python-decouple/
- **deep_translator**: Traducción automática.
  - https://pypi.org/project/deep-translator/
- **spacy**: procesamiento del lenguaje natural con capacidades avanzadas.
  - https://pypi.org/project/spacy/
- **sentence-transformers/all-mpnet-base-v2** este modelo convierte oraciones en vectores numéricos (embeddings) que capturan su significado semántico para tareas como búsqueda, comparación de similitud y agrupación de textos.
  - https://huggingface.co/sentence-transformers/all-mpnet-base-v2
- **facebook/bart-large-mnli** este modelo realiza tareas de clasificación de texto, especialmente clasificación zero-shot, determinando la probabilidad de pertenencia de un texto a varias categorías sin necesidad de entrenamiento específico en esas categorías.
  - https://huggingface.co/facebook/bart-large-mnli
- **all-MiniLM-L6-v2** este modelo genera embeddings de oraciones que capturan su significado semántico para tareas como búsqueda, comparación de similitud y agrupación de textos, optimizando velocidad y precisión.
  - https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Ejercicio 2:

Concepto de Rerank

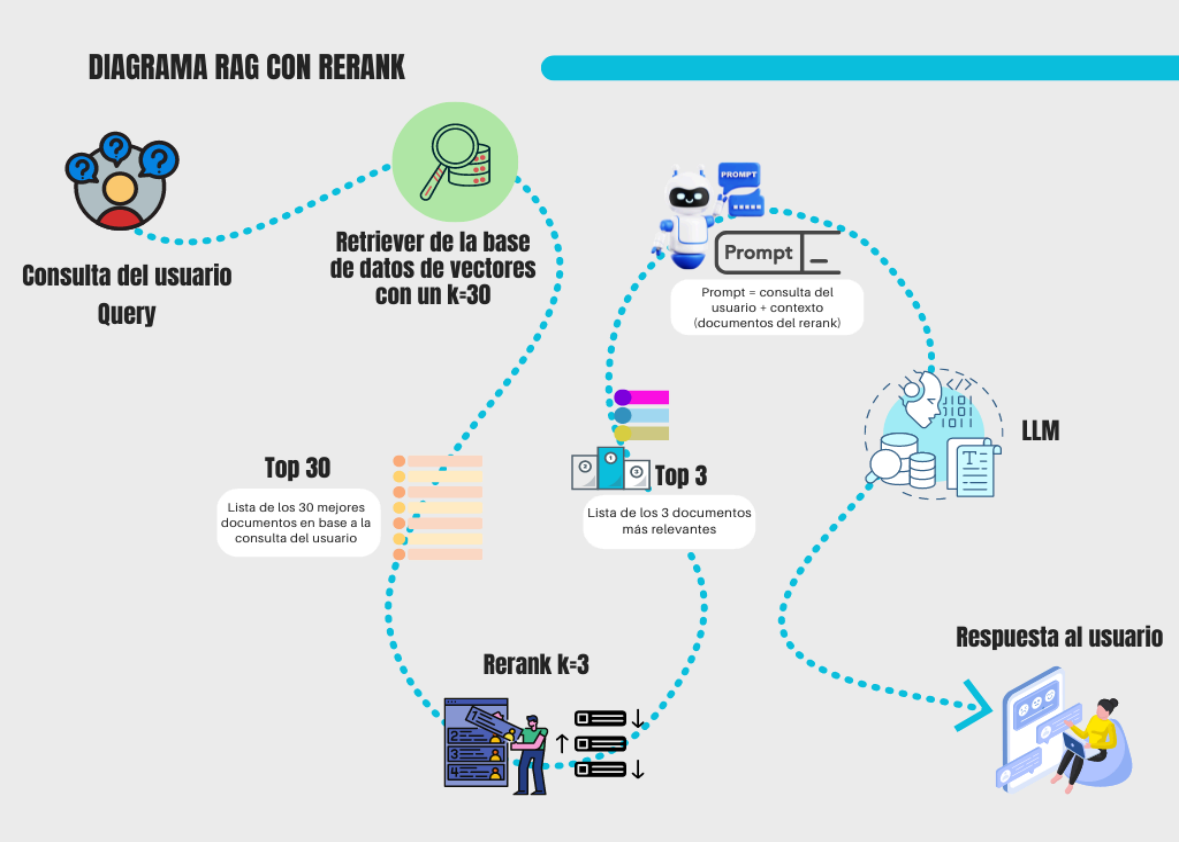
Es una técnica que se utiliza para mejorar la relevancia de los resultados de búsqueda en RAG. Consiste en tomar una lista inicial de documentos recuperados basados en una consulta y reordenarlos para asegurar que los más relevantes y útiles se presenten en las primeras posiciones. Es decir, cuando se realiza una búsqueda, el sistema primero recupera una lista de documentos que podrían ser relevantes. Luego, Rerank toma esta lista inicial y reordena los documentos para que los más útiles y precisos aparezcan primero.

Impacto en el desempeño de la aplicación

El Rerank mejora significativamente la precisión de los resultados. Al colocar los documentos más relevantes en las primeras posiciones, los usuarios obtienen respuestas más útiles y adecuadas a sus consultas. Esto también ayuda a eliminar información irrelevante, haciendo que las respuestas sean más claras y precisas.

Además, aunque Rerank requiere más recursos computacionales debido al uso de modelos más complejos y tiempo de procesamiento adicional, los beneficios en términos de relevancia y precisión generalmente justifican este costo.

Diagrama



2) ¿En qué Sección de su Código lo Aplicaría?

Tal como se puede observar en el diagrama que realicé para el ítem anterior, aplicaría el rerank después de realizar la recuperación de documentos (retriever) y antes de generar el prompt para la respuesta final (LLM). De esta manera, los documentos utilizados por el modelo de lenguaje serán los más relevantes y adecuados para la consulta del usuario. Este enfoque asegura que la respuesta generada esté basada en la información más precisa y contextualmente relevante, mejorando así la calidad y utilidad de las respuestas proporcionadas.

Fuentes:

Rerankers and Two-Stage Retrieval - Pinecone

https://www.pinecone.io/learn/series/rag/rerankers/

Mejorando los sistemas avanzados de RAG utilizando reranking con LangChain – MYSCALE

https://myscale.com/blog/es/maximizing-advanced-rag-models-langchain-reranking-techniques/

How to Build a RAG-Powered Chatbot with Chat, Embed, and Rerank - Cohere

https://cohere.com/blog/rag-chatbot#reranking

Rerank on LangChain - Cohere

https://docs.cohere.com/docs/rerank-on-langchain