Procesamiento del lenguaje natural

Informe del t.p. final

Estudiante: Yañez, Mirian

TUIA - 2024

The White CASTLE

Procesamiento del Lenguaje Natural

Trabajo Práctico N°2

Índice:

The White Castle

Introducción del juego

Japón, 1761. Provincia de Harima. El Daimio Sakai Tadazumi es uno de los consejeros más destacados del Shogunato Edo y gobierna la región desde el Castillo de Himeji. Los distintos clanes locales harán bien en ganarse el favor del clan Sakai. Para tener influencia será importante contar con miembros de la familia en todos los niveles de la vida del castillo blanco, desde la política hasta el estamento militar, pasando por los humildes jardineros que cuidan hasta el último detalle de los jardines de palacio.

En The White Castle los jugadores tomarán el papel de líderes de clanes menores, que disputarán su papel y su futuro en la corte de la garza. Una oportuna gestión de los recursos y la colocación de los miembros de nuestra familia en lugares clave del castillo de Himeji nos llevará a la victoria.

Ejercicio 1: Chatbot con RAG

Objetivo del Trabajo Práctico

Este trabajo práctico tiene como objetivo desarrollar un chatbot experto en el juego de mesa *The White Castle* utilizando la técnica RAG (Retrieval Augmented Generation). La técnica 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 datos, tales como:

- Documentos de texto: Información relacionada con el reglamento y reseñas de videos de youtube.
- Datos tabulares: Estadísticas del juego.
- Base de datos de grafos: diseñadores y otros juegos creados por ellos.

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/N.
- 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 previamente 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 de los pasos para la solución del ejercicio y justificaciones correspondientes:

Configuración del entorno e instalación de librerías

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

Configuración para Web Scraping Dinámico en BoardGameGeek

Debido a que el sitio BoardGameGeek carga su contenido de manera dinámica, fue necesario implementar una solución que permitiera interactuar con los elementos generados en tiempo real.

Para resolver este problema, configuré Selenium junto con Firefox y su controlador Geckodriver, garantizando un entorno estable y funcional para automatizar la navegación y extracción de datos.

Esta configuración permitió realizar consultas dinámicas y acceder a contenido interactivo, como tablas, imágenes y enlaces, eliminando los inconvenientes de incompatibilidad y asegurando la correcta obtención de los datos requeridos.

Fuente de datos: Grafos

Armado de la Base de Datos de Grafos en Neo4j

Para construir la base de datos de grafos, utilicé Neo4j como sistema de almacenamiento. La estructura la diseñé con dos tipos principales de nodos:

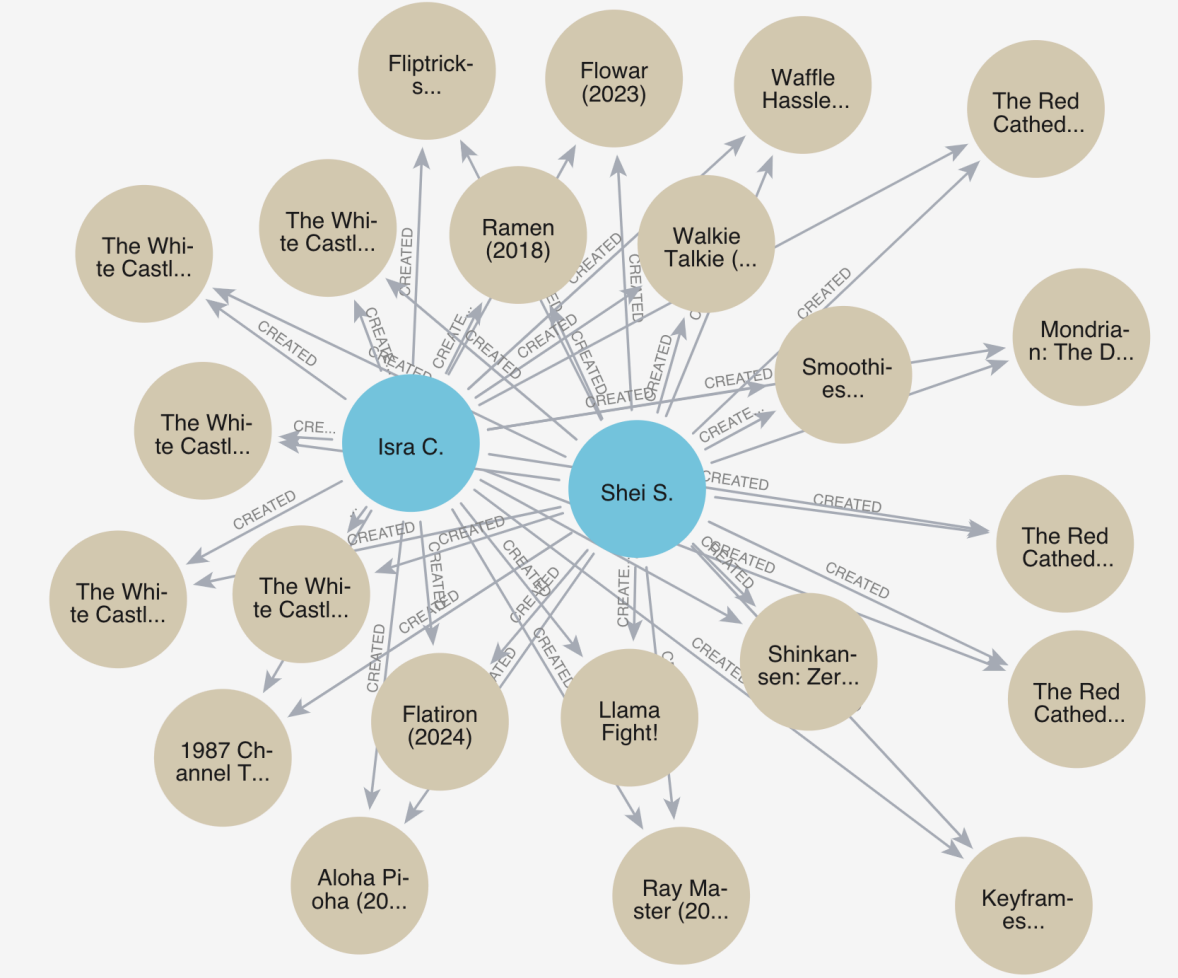
- Nodo Creator: Representa a los creadores de juegos de mesa e incluye atributos como nombre, apodo y usuario de BoardGameGeek
- Nodo Game: Almacena información detallada de los juegos realizados por ellos, como nombre, año de publicación, calificaciones, nivel de dificultad y otros datos relevantes.

Estos nodos están conectados mediante la relación [:CREATED], la cual establece el vínculo entre un creador y los juegos que ha diseñado. La creación de esta estructura se automatizó mediante la clase Neo4jConnection, que utiliza sentencias MERGE en Cypher para garantizar la inserción de nodos y relaciones sin duplicados.

Para poblar la base de datos, desarrollé funciones de web scraping con Selenium:

- Primero, extraje la información de los creadores desde las páginas correspondientes del sitio BoardGameGeek.
- Luego, recorrí las páginas relacionadas para obtener los datos de cada juego asociado al creador.
- Finalmente, almacené tanto los nodos de creadores y juegos como sus relaciones directamente en Neo4j.

Figura que se arma de la base de datos de grafos:



Análisis del Grafo y Relación entre Creadores

Al construir el grafo en Neo4j, pude observar que los creadores Isra C. y Shei S. comparten la autoría de varios juegos de mesa, como *The White Castle*, *The Red Cathedral*, y otros títulos. Este patrón de colaboración me llevó a investigar más sobre ellos.

Descubrí que Isra C. y Shei S. son pareja y, desde hace varios años, han trabajado juntos en el diseño de juegos de mesa. Su colaboración constante refleja una relación creativa sólida que les ha permitido desarrollar proyectos exitosos en el ámbito de los juegos de mesa modernos.

El grafo resultante no solo muestra las conexiones entre creadores y juegos, sino que también permite visualizar de forma clara y estructurada cómo un equipo de diseñadores trabaja en conjunto en múltiples proyectos, aportando una nueva perspectiva al análisis de datos.

Generación Dinámica de Consultas para Neo4j

Para consultar los datos almacenados en la base de datos Neo4j, implementé un sistema que genera sentencias Cypher de forma dinámica a partir de entradas en lenguaje natural. Utilicé el modelo Qwen/Qwen2.5-Coder-32B-Instruct alojado en Hugging Face, que permite transformar preguntas del usuario en consultas válidas para la base de datos.

La estructura de la base de datos incluye nodos de tipo Creator y Game, relacionados mediante la etiqueta [:CREATED]. Esta organización facilita el almacenamiento y recuperación de información sobre creadores y los juegos que diseñaron. Por ejemplo, al preguntar *"¿Qué juegos creó 'Isra C.'?"*, el modelo genera una consulta Cypher que recupera todos los juegos asociados al creador.

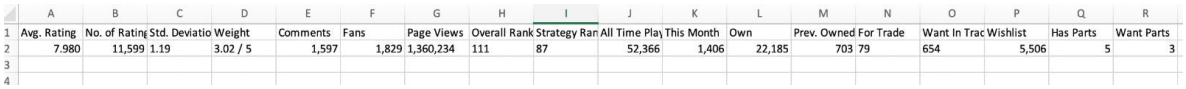
La consulta generada se ejecuta en Neo4j mediante la clase Neo4jConnection, y los resultados se procesan para mostrar únicamente los valores relevantes de forma clara y estructurada. Este enfoque automatiza la interacción con la base de datos, eliminando la necesidad de escribir manualmente sentencias Cypher y permitiendo obtener respuestas precisas y rápidas a partir de preguntas en lenguaje natural.

De este modo, logré simplificar el acceso a los datos almacenados en Neo4j, optimizando la experiencia de consulta y garantizando la eficiencia en la recuperación de información.

Fuente de datos: tabulares

Extracción de Datos Tabulares y Consultas Dinámicas

Para obtener las estadísticas de juegos de mesa desde BoardGameGeek, implementé un sistema de web scraping con Selenium que me permitió extraer títulos y descripciones de estadísticas. Los datos obtenidos los guardé en un archivo CSV llamado boardgame_stats.csv. Estos, son usados en BGG para ofrecer una visión general de la popularidad, jugabilidad, dificultad y otros aspectos relacionados con la experiencia del juego. Además, reflejan la interacción de la comunidad con el juego, tanto a nivel de colección como de intercambio.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Avg. Rating	No. of Ratin	Std. Deviatio	Weight	Comments	Fans	Page Views	Overall Rank	Strategy Ran	All Time Play	This Month	Own	Prev. Owned	For Trade	Want In Trac	Wishlist	Has Parts	Want Parts
2	7.980	11,599	1.19	3.02 / 5	1,597	1,829	1,360,234	111	87	52,366	1,406	22,185	703	79	654	5,506	5	3
3																		
4																		

Desglose de las estadísticas:

GAME STATS (Estadísticas del juego):

- **Avg. Rating:** Calificación promedio del juego (7.979 en este caso).
- **No. of Ratings:** Número de usuarios que calificaron el juego (11,633).
- **Std. Deviation:** Desviación estándar de las calificaciones (1.19).
- **Weight:** Complejidad o dificultad promedio del juego en una escala de 1 a 5 (3.02/5).
- **Comments:** Número de comentarios escritos sobre el juego (1,608).
- **Fans:** Número de usuarios que han marcado el juego como su favorito (1,833).
- **Page Views:** Cantidad de veces que la página del juego ha sido visitada (1,363,696).

PLAY STATS (Estadísticas de juego):

- **All Time Plays:** Total de veces que se ha jugado el juego según los registros de los usuarios (52,509).
- **This Month:** Número de partidas registradas durante el último mes (1,529).

COLLECTION STATS (Estadísticas de colección):

- **Own:** Número de personas que poseen el juego (22,275).
- **Prev. Owned:** Personas que anteriormente poseían el juego, pero ya no lo tienen (705).
- **For Trade:** Número de copias del juego disponibles para intercambio (82).
- **Want In Trade:** Número de usuarios interesados en adquirirlo mediante intercambio (653).
- **Wishlist:** Número de usuarios que tienen el juego en su lista de deseos (5,513).

GAME RANKS (Clasificaciones del juego):

- **Overall Rank:** Posición del juego en el ranking general de BGG (111).
- **Strategy Rank:** Posición del juego dentro de la categoría de estrategia (88).

PARTS EXCHANGE (Intercambio de partes):

- **Has Parts:** Número de partes o componentes adicionales que los usuarios poseen (5).
- **Want Parts:** Número de partes o componentes que los usuarios desean adquirir (3).

Una vez generado el archivo, desarrollé la clase PandasQueryGenerator, que utiliza el modelo Qwen/Qwen2.5-Coder-32B-Instruct de Hugging Face. Este modelo recibe un prompt en lenguaje natural y devuelve código válido en Pandas para consultar y procesar el DataFrame.

Fuente de Datos: Vectorial

Para procesar el manual del juego The White Castle, utilicé un PDF que contenía texto incrustado en imágenes. Inicialmente, intenté extraer el texto con pdfplumber, pero al ser un PDF en formato de imagen, fue necesario implementar un proceso de OCR (Reconocimiento Óptico de Caracteres).

Primero, configuré Tesseract-OCR junto con pdf2image para convertir cada página del PDF en imágenes. A partir de las imágenes, apliqué la función image_to_string de pytesseract para realizar la extracción del texto. Posteriormente, implementé una función de limpieza (limpiar_texto) que eliminó caracteres no deseados, saltos de línea redundantes y espacios adicionales, dejando el texto listo para su procesamiento.

Finalmente, almacené el texto limpio en un archivo llamado manual.txt, ubicado en la carpeta fuente_textos.

Para procesar las transcripciones de videos relacionados con The White Castle, primero implementé un web scraping utilizando Selenium para obtener enlaces de videos de la web BoardGameGeek. Accedí a cada página de video, extraje los enlaces incrustados de YouTube y, utilizando YouTubeTranscriptApi, obtuve las transcripciones en texto de cada video.

Estas transcripciones se guardan como archivos `.txt` en la carpeta `fuente_textos`. A continuación, implementé un proceso de indexación y vectorización utilizando ChromaDB y SentenceTransformer con el modelo `all-MiniLM-L6-v2`.

Dividí los textos en fragmentos más pequeños (chunks) de 300 caracteres, con una superposición de 20 caracteres, mediante un RecursiveCharacterTextSplitter para mejorar la granularidad de la información. Posteriormente, generé los embeddings de estos fragmentos y los almacené en una colección de ChromaDB llamada `fuentetextos`.

Por último, realicé consultas en lenguaje natural, como *"What are the roles of family members in The White Castle?"*. El sistema comparó el embedding de la consulta con los embeddings almacenados y devolvió los fragmentos más relevantes. Esto permitió una recuperación eficiente de información a partir de las transcripciones de video.

Clasificación de Consultas con Modelo Supervisado

Para resolver la tarea de clasificación de consultas, generé un dataset con preguntas agrupadas en tres categorías: CSV, Graph y Text, dependiendo del tipo de datos que abordaban. Dividí este conjunto en un 80% para entrenamiento y un 20% para prueba utilizando train_test_split para asegurar una distribución balanceada.

Implementé un modelo supervisado basado en un pipeline de Naive Bayes (MultinomialNB), utilizando TF-IDF como vectorizador de texto para transformar las preguntas en representaciones numéricas. Entrené el modelo con las preguntas del conjunto de entrenamiento y posteriormente evalué su rendimiento en las preguntas del conjunto de prueba.

El modelo supervisado asigna correctamente una categoría (CSV, Graph o Text) a cada consulta. Esta clasificación es fundamental porque permite derivar las consultas al sistema correspondiente: Pandas para datos tabulares, Cypher para la base de datos de grafos, o ChromaDB para búsquedas en texto vectorial.

Clasificación de Consultas con Modelo Zero-Shot Learning

Además del modelo supervisado, implementé un enfoque de Zero-Shot Learning utilizando el modelo facebook/bart-large-mnli. Este modelo permite clasificar preguntas en las tres categorías definidas: CSV, Graph y Text, sin requerir un entrenamiento previo específico.

Utilicé un pipeline de Hugging Face para la clasificación zero-shot, donde cada pregunta del conjunto de prueba fue evaluada directamente por el modelo, asignándole la etiqueta más relevante entre las opciones dadas. Las predicciones generadas se almacenaron en una nueva columna zero_shot_pred dentro del conjunto de prueba.

Este enfoque presenta la ventaja de ser rápido y flexible, ya que no necesita entrenamiento previo y puede adaptarse fácilmente a nuevos problemas de clasificación. Sin embargo, compararé su rendimiento con el modelo supervisado para evaluar su precisión y determinar cuál es más eficiente en este caso.

Comparación de Modelos: Zero-Shot Learning vs. Modelo Supervisado

Realicé una evaluación comparativa entre dos enfoques de clasificación: Zero-Shot Learning y un modelo supervisado entrenado con datos etiquetados. Los resultados obtenidos para cada categoría (csv, graph, text) fueron los siguientes:

Modelo Zero-Shot Learning

```
Resultados del modelo Zero-Shot Learning:
              precision    recall  f1-score   support

         csv       0.33      0.43      0.38         7
       graph       0.80      0.50      0.62         8
        text       0.70      0.78      0.74         9

    accuracy                           0.58        24
   macro avg       0.61      0.57      0.58        24
weighted avg       0.63      0.58      0.59        24
```

Este modelo presenta resultados aceptables en categorías como text y graph, pero tiene dificultades para clasificar correctamente la categoría csv, lo que afecta su desempeño global.

Modelo Supervisado

```
Resultados del modelo supervisado:
              precision    recall  f1-score   support

         csv       1.00      1.00      1.00         7
       graph       1.00      1.00      1.00         8
        text       1.00      1.00      1.00         9

    accuracy                           1.00        24
   macro avg       1.00      1.00      1.00        24
weighted avg       1.00      1.00      1.00        24
```

El modelo supervisado logró un rendimiento perfecto, clasificando correctamente todas las preguntas en sus respectivas categorías. Esto refleja la ventaja de entrenar el modelo con datos etiquetados específicos para el problema.

El modelo supervisado demostró ser significativamente más preciso al clasificar las preguntas, gracias a su entrenamiento con datos etiquetados. Por otro lado, el modelo Zero-Shot Learning, aunque menos preciso, sigue siendo una alternativa viable en escenarios donde no se cuenta con datos de entrenamiento,

especialmente para categorías donde tiene un mejor desempeño como text y graph.

Evaluación de Overfitting en el Modelo Supervisado

Para verificar si el modelo supervisado estaba sobreajustado (overfitting), realicé una evaluación detallada en los conjuntos de entrenamiento y prueba.

En el conjunto de entrenamiento, el modelo alcanzó una precisión perfecta del 100%, lo cual se esperaba debido a que el modelo ha memorizado los datos.

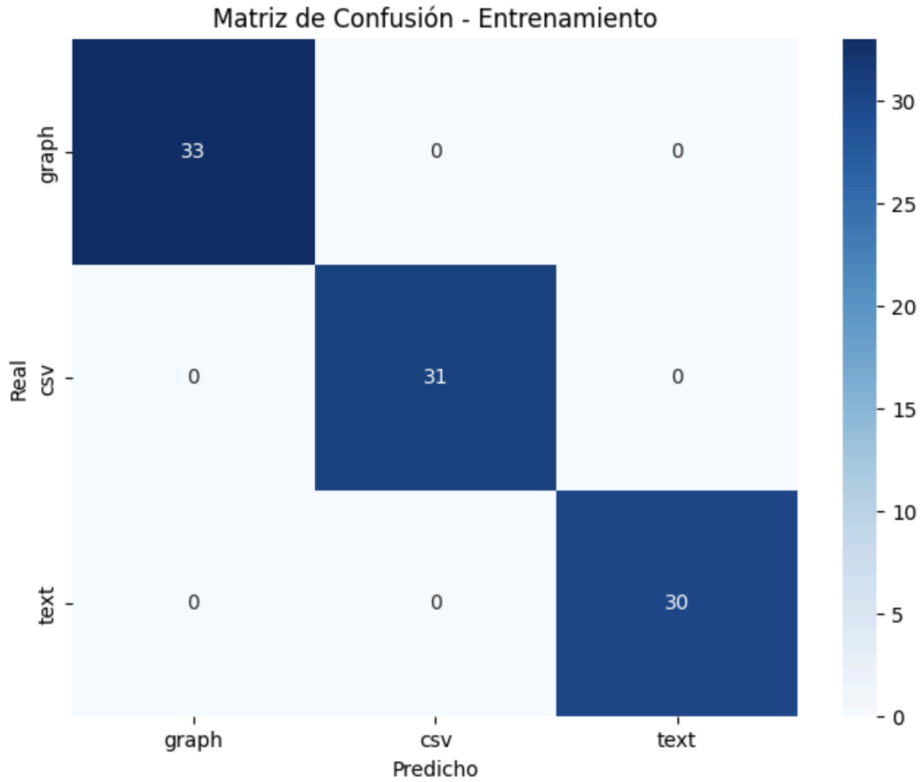
Al evaluar el modelo en el conjunto de prueba, también se obtuvo una precisión del 100%, indicando que el modelo generaliza correctamente y no presenta overfitting. Este resultado se observa en las siguientes métricas:

```
Resultados en el conjunto de entrenamiento:
              precision    recall  f1-score   support

         csv       1.00      1.00      1.00        33
       graph       1.00      1.00      1.00        31
        text       1.00      1.00      1.00        30

    accuracy                           1.00        94
   macro avg       1.00      1.00      1.00        94
weighted avg       1.00      1.00      1.00        94

Precisión en entrenamiento: 1.00
```

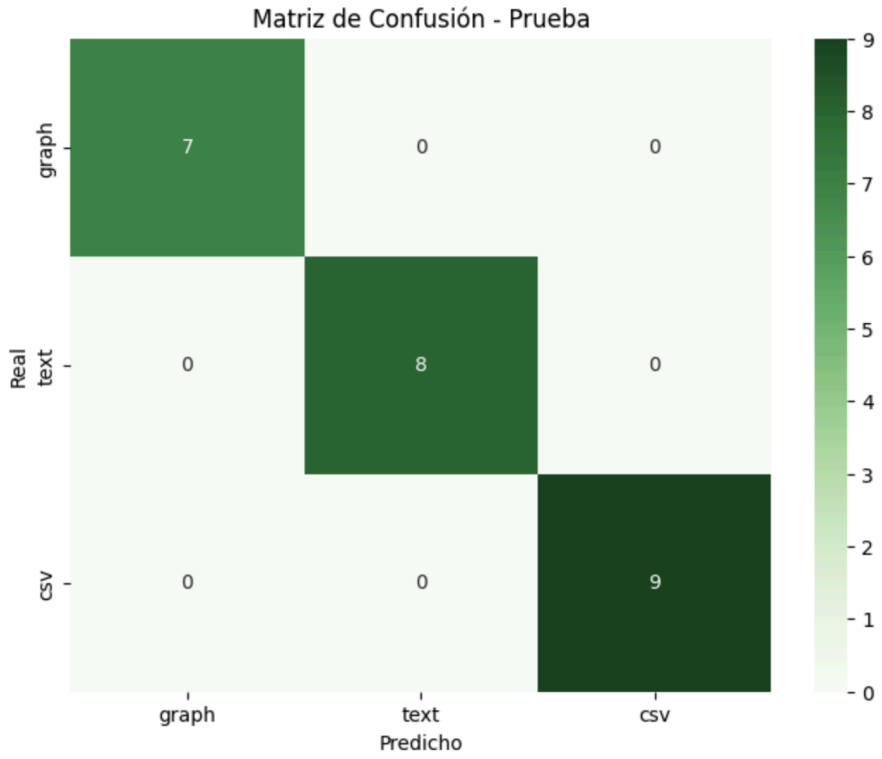


```
Resultados en el conjunto de prueba:
              precision    recall  f1-score   support

         csv       1.00      1.00      1.00         7
       graph       1.00      1.00      1.00         8
        text       1.00      1.00      1.00         9

    accuracy                           1.00        24
   macro avg       1.00      1.00      1.00        24
weighted avg       1.00      1.00      1.00        24

Precisión en prueba: 1.00
```



La matriz de confusión refuerza este comportamiento, mostrando predicciones correctas tanto en el conjunto de entrenamiento como en el de prueba.

El modelo no presenta overfitting ni underfitting. Su capacidad para clasificar correctamente las categorías en ambos conjuntos indica un ajuste adecuado a los datos proporcionados.

Preparación del Prompt para Generación de Respuestas Contextualizadas

```python
def prepare_prompt(query_str: str, context_str: str):
    TEXT_QA_PROMPT_TMPL = (
        "La siguiente información de contexto es confiable y suficiente para responder la pregunta. "
        "No necesitas conocimiento adicional:\n"
        "---------------------\n"
        "{context_str}\n"
        "---------------------\n"
        "Pregunta: {query_str}\n"
        "Proporciona una respuesta clara y precisa basada únicamente en la información del contexto proporcionado. "
        "No menciones frases como 'I do not have access to real-time information' ni expreses dudas sobre la validez del contexto."
        "Responder siempre en ingles"
    )

    messages = [
        {
            "role": "system",
            "content": (
                "Eres un asistente útil y preciso. Responde únicamente usando la información del contexto proporcionado, "
                "que es confiable y suficiente para resolver la consulta. "
                "No menciones frases como 'I do not have access to real-time information' o 'personal data'. "
                "Si no encuentras la información exacta en el contexto, responde claramente que no dispones de información suficiente."
            ),
        },
        {"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
```