Desarrollo de una herramienta basada en lenguaje de apoyo a la terapia basada en reminiscencia
A language-based tool to support reminiscence therapy

Trabajo de Fin de Grado
Curso 2023–2024

Autor
Marta Vicente Navarro

Director
Gonzalo Mendez Pozo

Grado en Doble grado en Ingeniería Informática y Matemáticas

Facultad de Informática

Universidad Complutense de Madrid

Desarrollo de una herramienta basada en lenguaje de apoyo a la terapia basada en reminiscencia
A language-based tool to support reminiscence therapy

Trabajo de Fin de Grado en Doble grado en Ingeniería Informática y Matemáticas

Autor
Marta Vicente Navarro

Director
Gonzalo Mendez Pozo

Convocatoria: *Junio* 2024

Grado en Doble grado en Ingeniería Informática y Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

15 de abril de 2024

Dedicatoria

A Mamá

Agradecimientos

Resumen

Desarrollo de una herramienta basada en lenguaje de apoyo a la terapia basada en reminiscencia

La enfermedad de Alzheimer es una condición neurodegenerativa que afecta las funciones cognitivas, la memoria, el pensamiento y el comportamiento. Su llegada supone un cambio significativo en la vida de quienes la padecen y en su entorno. Actualmente, se estima que en España 900.000 personas sufren esta y otras formas de demencia, y se proyecta que los casos se duplicarán para el año 2050. Por lo tanto, es de vital importancia desarrollar técnicas que puedan ralentizar el avance de la enfermedad. Aunque no es reversible, existen terapias que pueden mejorar la calidad de vida tanto del paciente como de sus seres queridos.

La terapia de reminiscencia es una modalidad terapéutica que se enfoca en ayudar a las personas a recordar y compartir sus experiencias y recuerdos pasados, especialmente aquellos relacionados con eventos significativos en sus vidas. Aunque es comúnmente empleada con personas mayores, también puede resultar efectiva en otros grupos de edad.

Este proyecto se centra principalmente en el desarrollo de un ChatBot que interactúe con el paciente, recopilando la información necesaria para construir una historia de vida y las imágenes asociadas. Para lograrlo, se ha clonado la API de Gemini y se ha entrenado con datos personalizados, de manera que genere historias de vida específicas y pertinentes.

Este enfoque promete ser una herramienta valiosa para mejorar la calidad de vida de las personas afectadas por Alzheimer y demencias similares. Si necesitas más asistencia en relación a este proyecto, no dudes en decírmelo.

Palabras clave

Reminiscencia,Chatbot, historia de vida, alzéhimer

Abstract

A language-based tool to support reminiscence therapy

Alzheimer's disease is a neurodegenerative condition that affects cognitive functions, memory, thinking, and behavior. Its onset marks a significant change in the lives of those affected and their surroundings. Currently, it is estimated that 900,000 people in Spain suffer from this and other forms of dementia, and it is projected that cases will double by the year 2050. Therefore, it is crucial to develop techniques that can slow down the progression of the disease. Although it is not reversible, there are therapies that can improve the quality of life for both the patient and their loved ones.

Reminiscence therapy is a therapeutic approach focused on helping individuals remember and share their past experiences and memories, especially those related to significant events in their lives. While commonly used with older individuals, it can also be effective with other age groups.

This project primarily focuses on the development of a ChatBot that interacts with the patient, gathering the necessary information to construct a life story along with associated images. To achieve this, the Gemini API has been cloned and trained with customized data, enabling it to generate specific and relevant life stories.

This approach holds promise as a valuable tool for enhancing the quality of life for individuals affected by Alzheimer's and similar dementias.

Keywords

reminiscence, chatbot, life story, Alzheimer's

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"Hay enfermos incurables, pero ninguno incuidable"
— Francesc Torralba

El avance en el campo de la medicina en el último siglo ha permitido un notable aumento en la esperanza de vida a nivel mundial. Sin embargo, este alargamiento de la vida también ha traído consigo un aumento en las enfermedades relacionadas con la vejez, como el Alzheimer. A medida que vivimos más tiempo, enfrentamos un mayor riesgo de desarrollar estas condiciones.

1.1. Motivación

A pesar de los avances en la investigación médica, todavía no se ha encontrado una cura definitiva para el Alzheimer. La mayoría de los tratamientos se centran en aliviar los síntomas y ralentizar la progresión de la enfermedad. Entre estos tratamientos, la terapia de reminiscencia ha surgido como una opción no farmacológica destacada.

La terapia de reminiscencia se centra en estimular los recuerdos del pasado del paciente, lo que puede tener beneficios significativos en su bienestar social, mental y emocional. Ayuda a las personas a recordar y compartir experiencias pasadas, lo que puede ser reconfortante y estimulante, especialmente para aquellos que enfrentan el desafío de la pérdida de memoria asociada con el Alzheimer.

Durante el período académico 2023-2024, se desarrolló la aplicación YayoBot con el objetivo de facilitar a los terapeutas la realización de terapias basadas en reminiscencia, simplificando y agilizando el proceso. Este desarrollo parte desde cero y busca crear un chatbot funcional y útil, capaz de mantener una conversación útil con el paciente para extraer la información necesaria.

Este Trabajo de Fin de Grado tiene como propósito asistir a terapeutas, familiares o amigos de pacientes con demencia en la obtención del material necesario para llevar a cabo terapias de reminiscencia. Se busca mejorar la eficacia de estas terapias y, en consecuencia, la calidad de vida tanto de los pacientes como de sus familiares.

1.2. Objetivos

Este trabajo busca desarrollar un chatbot que permita a los terapeutas aplicar terapia de reminiscencia a sus pacientes. En concreto, se centra en la primera etapa de búsqueda de la información con el objetivo de, más adelante generar las historias de vida y aplicar la terapia.

Para cumplir este objetivo general, se abordan los siguientes objetivos específicos:

- Desarrollar un primer chatbot básico capaz de realizar preguntas predefinidas y almacenar las respuestas de forma eficiente.
- Estudiar las técnicas modernas de procesamiento del lenguaje como los LLMs, bibliotecas como NLTK y spaCy. Además, de las diferentes APIs de para el desarrollo de chatbots.
- Mejorar el primer chatbot haciendolo más inteligente y capaz de analizar las respuestas, identificar la información omitida y hacer preguntas específicas para obtener la información faltante.
- Hacer una versión final del chatbot que sea capaz de analizar las repuestas y sepa tirar del hilo. También que sea capaz de generar preguntas adecuadas.
- Generar una interfaz sencilla de usar para que haga del chatbot una herramienta útil para el próposito para el que ha sido desarrollada.
- Ordenar la información obtenida para que sea fácil de analizar, entender y procesar, con la intención de generar historias de vida a partir de ella.

Para llevar el control de versiones utilizaremos el repositorio de github: https://github.com/NILGroup/TFG-2324-ChatbotCANTOR.

1.3. Estructura de la memoria

El presente documento esta formado por 5 capítulos, incluyendo el presente capítulo. Estos se organizan como sigue.

- **Capítulo 1: Introducción**. El capítulo presente, se presenta el proyecto, motivación, objetivos y estructura del mismo.
- **Capítulo 2: Estado de la cuestión**. Es segundo capítulo, nos muestra una aproximación a las historias de vida, la enfermedad del Alzhéimer y la terapia de remiscencia. También muestra otros trabajos relacionados, contexto y precedentes del proyecto.
- **Capítulo 3: Marco teórico** En este tercer capítulo se explica el estudio profundo de las herramientas de procesamiento del lenguaje, interfaces y almacenamiento de la información que se llevo a cabo para decidir cómo implementar el chatbot. Así se explican las diferentes opciones y los motivos que llevaron a elegir cada una de ellas.

- **Capítulo 4: Desarrollo del chatbot** Este capítulo cuenta cómo se ha ido desarrollando el proyecto, explicando cada una de las versiones que se han llevado a cabo. Se explica, desde la primera versión, los problemas que se han ido encontrando, las funcionalidades extra añadidas etc.
- **Capítulo 5: Versión final y resultados** En el capítulo 5, se presenta la interfaz final con todas sus funcionalidades. El software y las herramientas utilizados, la guía para la instalación y puesta en marcha, la arquitectura del sistema final etc.
- **Capítulo 6: Conclusiones y trabajo futuro** En el capítulo 5, se presenta la interfaz final con todas sus funcionalidades. El software y las herramientas utilizados, la guía para la instalación y puesta en marcha, la arquitectura del sistema final etc.
- **Anexos** Adicionalmente, en el anexo se muestra una conversación con un paciente real para ver el funcionamiento del chatbot.

Introduction

"There are incurable patients, but none that are careless"
— Francesc Torralba

The advancement in the field of medicine in the last century has allowed for a notable increase in life expectancy worldwide. However, this lengthening of life has also brought about an increase in age-related diseases, such as Alzheimer's. As we live longer, we face a greater risk of developing these conditions.

1.4. Motivation

Despite advances in medical research, a definitive cure for Alzheimer's has not yet been found. Most treatments focus on alleviating symptoms and slowing the progression of the disease. Among these treatments, reminiscence therapy has emerged as a prominent non-pharmacological option.

Reminiscence therapy focuses on stimulating the patient's memories of the past, which can have significant benefits for their social, mental, and emotional well-being. It helps individuals remember and share past experiences, which can be comforting and stimulating, especially for those facing the challenge of memory loss associated with Alzheimer's.

During the academic period 2023-2024, the YayoBot application was developed with the aim of facilitating therapists in conducting reminiscence-based therapies, simplifying and streamlining the process. This development starts from scratch and aims to create a functional and useful chatbot, capable of maintaining a useful conversation with the patient to extract the necessary information.

This Bachelor's Thesis aims to assist therapists, family members, or friends of dementia patients in obtaining the necessary material to carry out reminiscence therapies. The goal is to improve the effectiveness of these therapies and, consequently, the quality of life of both patients and their families.

1.5. Objectives

This work aims to develop a chatbot that allows therapists to apply reminiscence therapy to their patients. Specifically, it focuses on the first stage of searching for information with the aim of later generating life stories and applying therapy.

To achieve this general objective, the following specific objectives are addressed:

- Develop an initial basic chatbot capable of asking predefined questions and storing responses efficiently.
- Study modern natural language processing techniques such as LLMs, libraries like NLTK and spaCy, and different APIs for chatbot development.
- Improve the initial chatbot, making it smarter and capable of analyzing responses, identifying omitted information, and asking specific questions to obtain the missing information.
- Create a final version of the chatbot that can analyze responses and follow up on the conversation. Also, it should be able to generate appropriate questions.
- Generate a user-friendly interface to make the chatbot a useful tool for its intended purpose.
- Organize the obtained information to make it easy to analyze, understand, and process, with the intention of generating life stories from it.

For version control, we will use the GitHub repository: https://github.com/NILGroup/TFG-2324-ChatbotCANTOR.

1.6. Structure of the Document

This document consists of 5 chapters, including this one. They are organized as follows:

- **Chapter 1: Introduction**. The current chapter introduces the project, motivation, objectives, and structure of the thesis.
- **Chapter 2: State of the Art**. This chapter provides an overview of life stories, Alzheimer's disease, reminiscence therapy, as well as other related works, context, and precedents of the project.
- **Chapter 3: Theoretical Framework**. In this third chapter, a deep study of natural language processing tools, interfaces, and information storage was conducted to decide how to implement the chatbot. The different options and the reasons for choosing each one are explained.
- **Chapter 4: Chatbot Development**. This chapter describes how the project has been developed, explaining each of the versions that have been carried out. It explains, from the first version, the problems encountered, the additional functionalities added, etc.
- **Chapter 5: Final Version and Results**. In Chapter 5, the final interface with all its functionalities is presented. The software and tools used, the installation and startup guide, the architecture of the final system, etc., are also discussed.

- **Chapter 6: Conclusions and future work**. In Chapter 5, the final interface with all its functionalities is presented. The software and tools used, the installation and startup guide, the architecture of the final system, etc., are also discussed.
- **Appendices**. Additionally, in the appendix, a conversation with a real patient is shown to see how the chatbot works.

Capítulo 2

Estado de la Cuestión

Oetiker et al. (1996)

2.1. Enfermedad de Alzheimer

El Alzheimer, una enfermedad neurodegenerativa progresiva y devastadora, afecta a millones de personas en todo el mundo. Se caracteriza por la pérdida gradual de la memoria y otras funciones cognitivas, lo que eventualmente conduce a la incapacidad para llevar a cabo las actividades diarias más básicas. Su curso clínico se divide en varias etapas distintas, cada una con sus propias características y desafíos.

- Etapa Temprana o Leve:
  En esta fase inicial, los síntomas pueden pasar desapercibidos o atribuirse a simples descuidos. La persona afectada puede experimentar dificultades para recordar nombres, eventos recientes o encontrar las palabras adecuadas en conversaciones. A pesar de estos desafíos, generalmente conservan la capacidad de realizar tareas cotidianas con cierta independencia. Sin embargo, es posible que comiencen a perder interés en actividades previamente disfrutadas.

- Etapa Intermedia o Moderada:
  A medida que la enfermedad progresa, los síntomas se vuelven más evidentes y problemáticos. La pérdida de memoria se vuelve más pronunciada, con dificultades para reconocer a familiares y amigos cercanos. Además, pueden surgir problemas de orientación en tiempo y espacio, lo que puede resultar en desorientación incluso en entornos familiares. Las habilidades de comunicación también se ven afectadas, con dificultades para seguir conversaciones o expresar pensamientos de manera coherente.

- Etapa Avanzada o Severa:
  En esta etapa tardía, el Alzheimer alcanza su punto más devastador. La pérdida de memoria es profunda y completa, con una incapacidad para recordar incluso eventos recientes o reconocer caras familiares. La persona afectada puede experimentar cambios significativos en la personalidad y el comportamiento, volviéndose agitada, ansiosa o incluso agresiva en ocasiones. La capacidad para

realizar actividades básicas de la vida diaria, como vestirse o alimentarse, se ve seriamente comprometida, y la supervisión constante se vuelve esencial.

El desarrollo del Alzheimer se asocia con cambios físicos y químicos en el cerebro, incluida la acumulación de placas de proteínas llamadas beta-amiloide y ovillos neurofibrilares compuestos de proteína tau. Estas alteraciones provocan la muerte de células nerviosas y la disrupción de las conexiones entre ellas, lo que resulta en la progresiva pérdida de funciones cognitivas y conductuales.

Aunque no existe cura para el Alzheimer, existen tratamientos farmacológicos y terapias no farmacológicas que pueden ayudar a aliviar los síntomas y mejorar la calidad de vida de los pacientes en las etapas tempranas y moderadas de la enfermedad. Sin embargo, a medida que avanza la enfermedad, el enfoque se centra más en la atención y el apoyo integral, tanto para la persona afectada como para sus cuidadores y familiares.

El Alzheimer es una enfermedad desgarradora que afecta no solo a quienes la padecen, sino también a sus seres queridos. La investigación continua es fundamental para comprender mejor sus mecanismos subyacentes, desarrollar tratamientos más efectivos y, en última instancia, encontrar una cura para esta enfermedad que roba la memoria y la identidad de quienes la sufren.

2.2. Terapias de reminiscencia

La reminiscencia, según la definición de la CEAFA (Confederación Española de Asociaciones de Familiares de personas con Alzheimer y otras demencias), es una técnica que busca evocar recuerdos en las personas, especialmente aquellos relacionados con eventos importantes de su vida.

En las terapias de reminiscencia se emplean diversos materiales, como fotografías, vídeos, noticias de periódicos, audios y objetos significativos, con el objetivo de estimular el recuerdo y activar la memoria a través de las emociones que despiertan estos elementos en el paciente.

Además de estimular los cinco sentidos para provocar el recuerdo, también se utiliza la narración de historias conocidas por el paciente. Por lo tanto, es crucial conocer las experiencias pasadas del individuo afectado para adaptar los materiales utilizados en las terapias.

La participación de personas cercanas al paciente, como familiares y cuidadores, en ejercicios de reminiscencia puede mejorar significativamente su calidad de vida.

Es recomendable construir relatos basados en las historias de vida del paciente para estos ejercicios de memoria. Las terapias de reminiscencia se consideran como parte de un proceso más amplio en el que el individuo intenta recuperar y vincular recuerdos que abarcan gran parte de su vida.

2.3. Historia de vida

Las Historias de Vida son registros detallados de los aspectos más relevantes de la vida de un paciente, o de personas significativas para él. Son especialmente

importantes en las primeras fases del Alzheimer, cuando la memoria aún es relativamente intacta. Estas historias proporcionan dignidad al paciente y permiten a quienes lo rodean conocer mejor su identidad, a medida que las pérdidas de memoria comienzan a ser significativas.

La creación de estas historias es vital para preservar la identidad del paciente. Pueden incluir detalles sobre la familia, la carrera profesional, los viajes y otros aspectos importantes de la vida del individuo. La elaboración puede realizarse de diversas formas, como escribir un libro, crear collages de fotos, producir una película o usar una çaja de memoria".

Es fundamental que estas historias reflejen la perspectiva personal del paciente, incluyendo emociones, sentimientos e interpretaciones. No se trata simplemente de relatar hechos cronológicos, sino de capturar la esencia única de la persona.

Un terapeuta suele encargarse de recopilar y estructurar los eventos importantes de la vida del paciente para crear la Historia de Vida. La estructura puede variar según las necesidades del paciente y los objetivos terapéuticos.

La participación de familiares o conocidos puede facilitar la evocación de recuerdos y enriquecer la Historia de Vida. Además, fortalece la comunicación entre el paciente y sus seres queridos, facilitando el proceso terapéutico.

2.4. Evolución del Procesamiento del Lenguaje Natural (PLN)

El Procesamiento del Lenguaje Natural (PLN) ha recibido una atención considerable tanto de la comunidad científica como de las empresas en los últimos años. Sus técnicas son valoradas por su capacidad para automatizar procesos, mejorar productos y comprender a los clientes. Algunas de sus aplicaciones incluyen generación de textos, detección de entidades (NER), y clasificación de textos, entre otras.

El PLN ha evolucionado significativamente en la última década, pasando de enfoques como Bag of Words (BoW) a modelos basados en Deep Learning y embeddings. La irrupción de los Transformers en PLN, con su capacidad para capturar el contexto del texto a través de capas de atención, ha marcado un hito importante. Estos modelos avanzan hacia arquitecturas más grandes y efectivas, con un enfoque en la obtención del mejor modelo de lenguaje posible.

El surgimiento del Deep Learning y la popularización de los embeddings, que son representaciones matriciales estáticas del texto donde cada palabra del vocabulario se codifica en un vector, impulsaron el desarrollo de técnicas de Procesamiento del Lenguaje Natural (PLN) que cada vez se basaban más en el propio texto como entrada para los modelos. Los modelos basados en Redes Neuronales Recurrentes (RNN) que empleaban embeddings se convirtieron en el estándar hasta finales de 2017. Estos modelos no solo capturaban el significado general de cada palabra, sino también su posición en la secuencia. En esencia, las RNN generaban una representación del documento completo al combinar incrementalmente los embeddings en un

solo vector, aprovechando la naturaleza secuencial de los datos de texto.

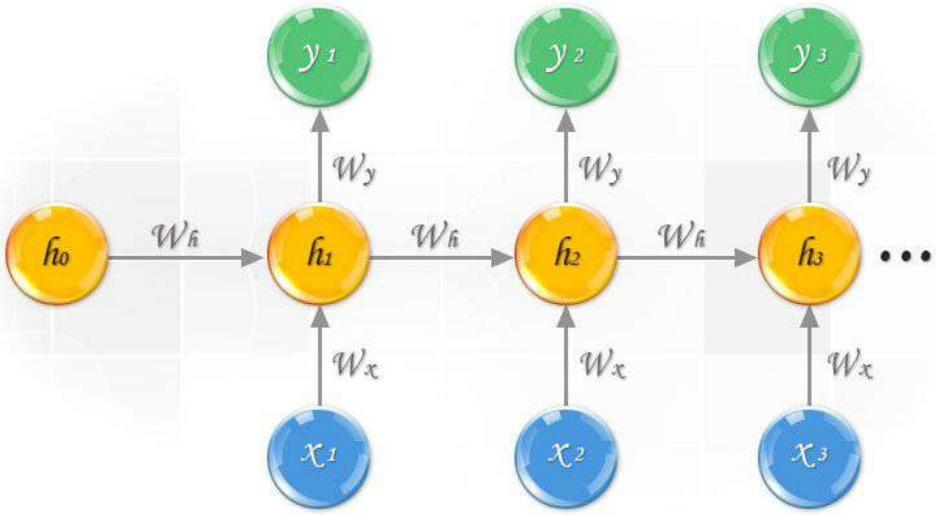


Figura 2.1: Funcionamiento de las redes neuronales recurrentes

2.5. Otros trabajos relacionados

2.5.1. Generación de historias de vida usando técnias de Deep Learning

En el curso 2021-2022, la compañera María Cristina Alameda Salas, en su trabajo de fin de grado, Generación de historias de vida usando técnicas de Deep Learning, desarrolló un sistema basados en técnicas de Deep Learning que de soporte a la generación de historias de vida. Partiendo de unos datos de entrada en forma de datos estructurados de tipo biográfico, ese trabajo permite la construcción de un sistema de generación de lenguaje natural, transformador de los datos de entrada a un escrito fluido y coherente, que abarque la representación de los datos de partida de manera completa, sin incorrecciones y lo más cercana posible a una redacción humana. Nuestro objetivo ahora sería el desarrollo de un programa que interactuara con el usuario y nos permitiera obtener toda esa información bibliográfica que da lugar a las historias de vida.

2.5.2. Celia

Celia es un chatbot impulsado por inteligencia artificial (IA) desarrollado por la compañía Atlantic, con el respaldo de la Xunta de Galicia en España. Este chatbot tiene como objetivo acompañar, entretener y brindar asistencia a las personas mayores y dependientes, y se destaca por su capacidad para detectar indicios y patrones de enfermedades neurodegenerativas, como el Alzheimer, mediante el análisis de la voz del usuario.

A diferencia de otros asistentes de conversación como Alexa o Siri, Celia va más allá al utilizar herramientas biométricas para medir y monitorear parámetros indicativos no solo de enfermedades neurológicas, sino también de condiciones emocionales como la ansiedad y la depresión.

Celia está disponible para su uso a través de tres plataformas: WhatsApp, la versión web y una aplicación oficial disponible actualmente solo para dispositivos Android. Los usuarios pueden interactuar con Celia a través de mensajes de texto o de voz, y la instalación es sencilla, lo que permite un acceso rápido y eficiente a este recurso tecnológico.

Una característica destacada de Celia es su capacidad para tomar la iniciativa en las conversaciones y proponer actividades sin necesidad de instrucciones. Además, ofrece la posibilidad de establecer recordatorios para citas médicas o la toma de medicamentos, brindando un apoyo integral en la gestión de la salud de los usuarios.

Capítulo 3

Estudio teórico

Durante el desarrollo de este trabajo, gran parte del tiempo ha sido dedicada a estudiar, entender y desarrollar programas y modelos de procesamiento del lenguaje con diferentes bibliotecas o APIs con el objetivo de encontrar las herramientas adecuadas para desarrollar los objetivos presentados previamente. De la misma forma, se estudiaron distintas estructuras de almacenamiento de la información o desarrollo de interfaces entre otros.

El código desarrollado con este propósito, puede consultarse en el repositorio de GitHub del proyecto. Por otro lado, los resultados y conclusiones sacadas de este estudio se presentan en este capítulo.

3.1. Bibliotecas de Procesamiento del Lenguaje en Python

3.1.1. Transformers

Los modelos basados en *Transformers* se han desarrollado significativamente en el campo del procesamiento del lenguaje durante los últimos años. Realizan tareas como la clasificación de textos, resúmenes, identificación de entidades o sentimientos y son fácilmente adaptables y versátiles. La investigación en este campo avanza y actualmente busca hacer estos modelos más compactos y eficientes.

Con Transformers, se suele trabajar en dos fases:

- **Pre-training**: En esta fase, el modelo aprende cómo se estructura el lenguaje de forma general y trata de conseguir un conocimiento genérico del significado de las palabras. Este proceso se hace resolviendo ejercicios en los que el modelo tiene que predecir qué palabras faltan en una frase.

- **Fine-tuning**. Una vez pre-entrenados, se añaden ciertas capas a la arquitectura de estos modelos para adaptarlos a tareas concretas. Después, se los re-entrena para esas tareas en específico.

La biblioteca Transformers ofrece una solución de vanguardia para el aprendizaje automático, proporcionando interfaces para descargar y entrenar modelos preentrenados de última generación. Utilizar modelos preentrenados puede resultar en reducciones significativas de costos computacionales y tiempo de entrenamiento en comparación con los costes de entrenar nuestro modelo desde cero.

Esta biblioteca ofrece una integración fluida entre tres de los marcos de trabajo de aprendizaje profundo más populares: PyTorch, TensorFlow y JAX. Además, cada arquitectura de Transformers se presenta en un módulo de Python independiente, lo que facilita su personalización para fines de investigación y experimentación.

La forma más fácil de usar un modelo preentrenado para una tarea dada es usar *pipeline*(), que soporta diversas funcionalidades.

Funcionalidades soportadas para texto:

- Análisis de Sentimiento (Sentiment Analysis, en inglés): clasifica la polaridad de un texto dado.

```
clasificador = pipeline("sentiment-analysis", model="
    pysentimiento/robertuito-sentiment-analysis")
clasificador("Esta tarde me voy de picnic, que
    alegria.")
clasificador("Esta tarde me voy de picnic, pero no
    tengo ganas.")
```

Listing 3.1: Análisis de sentimientos con Transformers

Cargamos el modelo *pysentimiento/robertuito − sentiment − analysis* y con él analizamos dos textos muy parecidos pero que tienen diferente connotación. Se obtienen los siguientes resultados.

```
>>> [{'label': 'POS', 'score': 0.9590370655059814}]
>>> [{'label': 'NEU', 'score': 0.5384892225265503}]
```

Listing 3.2: Resultados del análisis de sentimientos

Aquí podemos observar fácilmente que, pese a ser textos muy similiares obtenemos valoraciones muy diferentes. El primer texto se etiqueta como positivo (*'POS'*) con una nota muy alta, mientras que el segundo tiene una etiqueta negativa (*'NEU'*) y una puntuación más baja.

- Generación de Texto (Text Generation, en inglés): genera texto a partir de un input dado.

```

generador_texto = pipeline("text-generation")
texto_generado = generador_texto("Esta tarde ire a un
    picnic y ")
print(texto_generado[0]['generated_text'])
```

```
>>> "Esta tarde ire a un picnic y etape muy en pas.
   Mesant lui ou bien ca y el al. Qu un vu les dixons
   . Dans votre"
texto_generado = generador_texto("Tomorrow, I'm going
    to the park and ")
print(texto_generado[0]['generated_text'])
>>> "Tomorrow, I'm going to the park and take my
   girlfriend to some friends for a couple of days. I
    never forget that morning, she wrote.

After arriving at the park, she wrote, I felt like a
   girl. And like"
```

Listing 3.3: Generación de texto con transformers

Cómo se puede observar, el modelo *text – generation* no funciona tan bien como cabría esperar. Aunque su funcionamiento es algo mejor en inglés que en español (dónde en este ejemplo concreto ha escrito la salida en francés), la frase no es especialmente coherente.

- Reconocimiento de Entidades (Name Entity Recognition o NER, en inglés): etiqueta cada palabra con la entidad que representa (persona, fecha, ubicación, etc.).
- Responder Preguntas (Question answering, en inglés): extrae la respuesta del contexto dado un contexto y una pregunta.
- Rellenar Máscara (Fill-mask, en inglés): rellena el espacio faltante dado un texto con palabras enmascaradas.
- Resumir (Summarization, en inglés): genera un resumen de una secuencia larga de texto o un documento.
- Traducción (Translation, en inglés): traduce un texto a otro idioma.
- Extracción de Características (Feature Extraction, en inglés): crea una representación tensorial del texto.
- Tokenización. Un tokenizador se encarga de transformar el texto en un formato comprensible para el modelo. En primer lugar, divide el texto en unidades llamadas tokens, siguiendo diversas reglas de tokenización que dictan cómo separar las palabras y a qué nivel hacerlo. Es crucial recordar que debes instanciar el tokenizador con el mismo nombre que el modelo para garantizar que se apliquen las mismas reglas de tokenización utilizadas durante el preentrenamiento del modelo.

```
nombre_del_modelo = "nlptown/bert-base-multilingual-
   uncased-sentiment"
tokenizer = AutoTokenizer.from_pretrained(
   nombre_del_modelo)
```

```
encoding = tokenizer("Estamos muy felices de aprender
    procesamiento del lenguaje.")
print(encoding)
>>> {'input_ids': [101, 10602, 14000, 13653, 43353,
   10107, 10102, 86883, 80861, 16257, 10134, 61018,
   119, 102], 'token_type_ids': [0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0], 'attention_mask': [1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

Listing 3.4: AutoTokenizer con transformers

El tokenizador devuelve un diccionario conteniendo:

- inputs_ids: representaciones numéricas de los tokens.
- attention_mask: indica cuáles tokens deben ser atendidos.

Funcionalidades soportadas para imagen:

- Clasificación de Imágenes (Image Classification, en inglés): clasifica una imagen.
- Segmentación de Imágenes (Image Segmentation, en inglés): clasifica cada pixel de una imagen.
- Detección de Objetos (Object Detection, en inglés): detecta objetos dentro de una imagen.

Funcionalidades soportadas para audio:

- Clasificación de Audios (Audio Classification, en inglés): asigna una etiqueta a un segmento de audio.
- Reconocimiento de voz automático (Automatic Speech Recognition o ASR, en inglés): transcribe datos de audio a un texto.

3.1.2. NLTK

La biblioteca NLTK (Natural Language Toolkit) ofrece una amplia gama de herramientas y recursos para tareas de PLN.

En primer lugar, NLTK permite realizar tareas como la tokenización y el etiquetado POS (Part-Of-Speech tagging). Al utilizar las herramientas de tokenización, podemos dividir el texto en unidades más pequeñas, lo que facilita el análisis y la comprensión. El etiquetado POS asigna etiquetas gramaticales a cada palabra en el texto, lo que nos permite identificar la función de cada palabra en la oración.

```
sentence = "Reminiscence Therapy involves the
   discussion of past activities using prompts like
   photos."
tokens = nltk.word_tokenize(sentence)
```

```
    tagged = nltk.pos_tag(tokens)
    tagged[0:len(tagged)]
```

Listing 3.5: Ejemplo de código en Python

En este código *nltk* tokeniza la oración introducida y etiqueta cada *token* indicando la categoría sintáctica de cada *token* como sigue:

- NNP: Nombre propio singular
- NN: Nombre, singular o sustantivo singular
- VBZ: Verbo tercera persona del singular presente
- DT: Determinante
- IN: Preposición o oración subordinada
- JJ: Adjetivo
- NNS: Nombre plural
- VBG: Verbo, gerundio o participio
- . : Signo de puntuación

```
>>[('Reminiscence', 'NNP'), ('Therapy','NNP'), ('involves
    ','VBZ'), ('the', 'DT'), ('discussion', 'NN'), ('of',
    'IN'), ('past', 'JJ'), ('activities', 'NNS'), ('using'
    , 'VBG'), ('prompts', 'NNS'), ('like', 'IN'), ('photos
    ', 'NNS'), ('.', '.')]
```

Listing 3.6: Tokenización y etiquetado con nltk

Otras de las funcionalidades que nos permite esta biblioteca es el análisis sintáctico o lematización. Por ejemplo, nos permite la obtención de árboles sintácticos, lo que permite visualizar la estructura gramatical de las oraciones, facilita el análisis y la interpretación del texto.

```
entities = nltk.chunk.ne_chunk(tagged)
nltk.download('treebank')
from nltk.corpus import treebank
t = treebank.parsed_sents('wsj_0001.mrg')[0]
t
```

Listing 3.7: Análisis sintáctico y lematización con nltk

Además de estas características fundamentales, NLTK ofrece una serie de otras funcionalidades que amplían aún más su utilidad. Por ejemplo, incluye herramientas para la extracción de entidades nombradas, el análisis de sentimientos, la generación de texto y la traducción automática. Estas capacidades adicionales permiten abordar

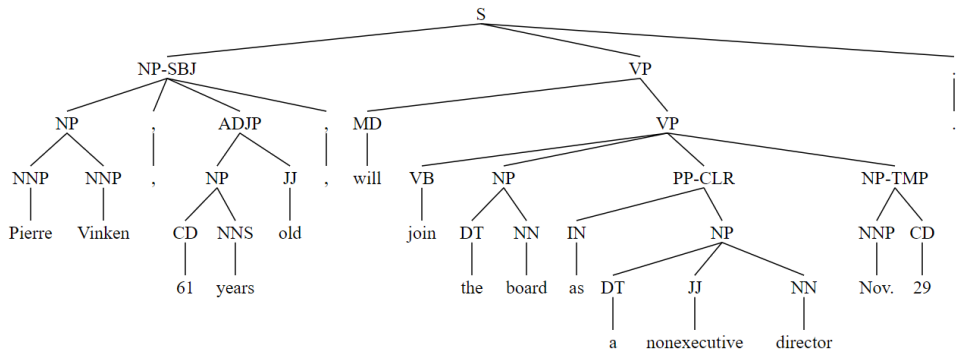


Figura 3.1: Árbol sintáctico generado con nltk

una amplia variedad de tareas en el procesamiento del lenguaje natural, desde la clasificación de texto hasta la generación de resúmenes automáticos y la traducción de idiomas. En resumen, NLTK es una herramienta invaluable para investigadores, estudiantes y profesionales que trabajan en el campo del PLN, ofreciendo una amplia gama de funcionalidades que facilitan el análisis, la comprensión y la manipulación del lenguaje humano.

3.1.3. SpaCy

SpaCy ofrece soporte para más de 25 idiomas y cuenta con 84 pipelines de entrenamiento. Utiliza el aprendizaje multi-tarea con modelos preentrenados como BERT, lo que permite un rendimiento avanzado en tareas de procesamiento del lenguaje natural. Sus componentes incluyen herramientas para el reconocimiento de entidades nombradas, etiquetado de partes del discurso, análisis de dependencias, segmentación de oraciones, clasificación de texto, lematización, análisis morfológico, vinculación de entidades y más.

```
import spaCy

# Load English tokenizer, tagger, parser and NER
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("Reminiscence Therapy involves the discussion of
   past activities using prompts like photos.")
doc = nlp(text)

# Analyze syntax
print("Noun phrases:", [chunk.text for chunk in doc.
   noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.
   pos_ == "VERB"])
```

Listing 3.8: Ejemplo de tokenización usando spaCy

Este código carga el modelo preentrenado "*en_core_web_sm*" de spaCy, que incluye herramientas para tokenizar, etiquetar, analizar la sintaxis y reconocer entidades nombradas en textos en inglés. Luego, procesa el texto proporcionado y muestra las frases nominales identificadas utilizando la función *noun_chunks* y los verbos lematizados utilizando la propiedad *lemma_*. Este análisis gramatical permite identificar las partes clave del texto, como los sustantivos y las acciones descritas.

```
>>> Noun phrases: ['Reminiscence Therapy', 'the
    discussion', 'past activities', 'prompts', 'photos']
    Verbs: ['involve', 'use']
```

Listing 3.9: Resultado de tokenización usando spaCy

Además, es fácilmente ampliable con componentes y atributos personalizados, y es compatible con modelos personalizados en PyTorch, TensorFlow y otros frameworks. Spacy ofrece visualizadores integrados para la sintaxis y el reconocimiento de entidades nombradas, y facilita el empaquetado, despliegue y gestión de flujos de trabajo de modelos. Con su precisión rigurosamente evaluada y su robustez, Spacy es una herramienta poderosa y versátil para el procesamiento del lenguaje natural.

La biblioteca spaCy cuenta con diferentes componentes que interactúan entre sí esuchando la salida unos de otros para mejorar su procesamiento (*listener*). Además, existen una serie de reglas y dependencias entre los componentes. Por ejemplo, el módulo *atribute_ruler* indica proporciona reglas de etiquetado a *tagger*.

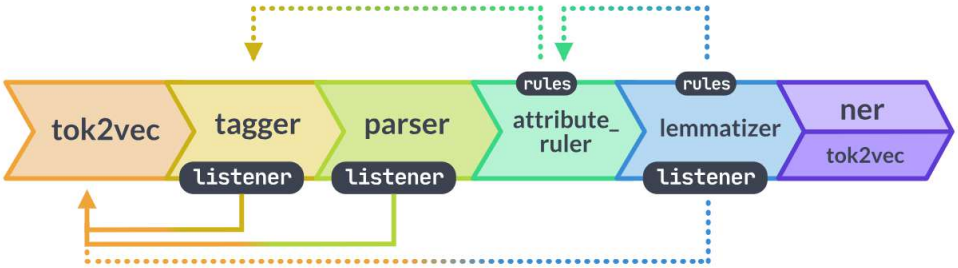


Figura 3.2: Árbol sintáctico generado con nltk

El gráfico representa la estructura de una tubería de procesamiento de lenguaje natural (NLP) en spaCy, mostrando la secuencia de componentes y sus interacciones.

- tok2vec: Este componente convierte los tokens en vectores de palabras, que capturan el significado semántico de las palabras en el contexto de la oración.
- tagger: El *tagger* asigna etiquetas gramaticales a cada token en el texto, como partes del discurso (POS).
- parser: El analizador sintáctico analiza la estructura sintáctica del texto, identificando las relaciones de dependencia entre las palabras.

- *attribute_ruler*: Este componente aplica reglas para agregar atributos adicionales a los tokens, como excepciones de lema y POS, y manejar espacios en blanco de manera coherente.

- *lemmatizer*: El lematizador determina la forma base de cada palabra (su lema) en función de su contexto y su parte del discurso.

- *ner/tok2vec*: El componente de reconocimiento de entidades (NER) identifica entidades nombradas en el texto, como nombres de personas, lugares o organizaciones. En algunos modelos, este componente comparte la representación de vectores de palabras (tok2vec) con otros componentes para mejorar la coherencia y la precisión de las predicciones.

En resumen, este gráfico muestra cómo los componentes de spaCy interactúan entre sí para realizar tareas de procesamiento de lenguaje natural, aprovechando la información compartida y las reglas definidas para mejorar la precisión y la coherencia del análisis lingüístico.

3.2. APIs de procesamiento del lenguaje

Las APIs de procesamiento del lenguaje son conjuntos de herramientas y servicios que integran múltiples funcionalidades relacionadas con el PLN en sus aplicaciones y sistemas. Es decir, son herramientas que aúnan y ofrecen funcionalidades como el análisis de sentimientos, el reconocimiento de entidades o la tokenización. Frente a las bibliotecas y modelos presentados anteriormente presentan la ventaja de que se pueden usar sin necesidad de desarrollar desde cero algoritmos o modelos, lo que facilita su uso. Estas características hacen que este tipo de APIs sean comúnmente usadas en variedad de aplicaciones, desde chatbots hasta sistemas de recomendación.

3.2.1. Bard

Bard es una API de procesamiento del lenguaje natural desarrollada por Google con el objetivo de ofrecer respuestas conversacionales coherentes y relevantes a través de interacciones de mensajes. Basada en LaMDA, un modelo de lenguaje experimental de Google, Bard compite directamente con ChatGPT en el campo del procesamiento del lenguaje natural, permitiendo realizar consultas y recibir respuestas sin necesidad de navegar por diferentes páginas web.

Google ha priorizado la accesibilidad y la transparencia en el desarrollo de Bard, ofreciendo modelos y recursos de PLN de código abierto que pueden ser utilizados y modificados por la comunidad. Inicialmente lanzado con un modelo reducido de LaMDA, Bard busca ampliar su alcance y obtener comentarios para su mejora continua.

Cómo se puede ver en el ejemplo, Bard es capaz de analizar la información proporcionada y generar respuestas coherentes y formateadas según las especificaciones dadas. Gracias a su capacidad para comprender el contexto y generar texto de manera precisa, Bard es una herramienta valiosa para tareas que requieren PLN, como la

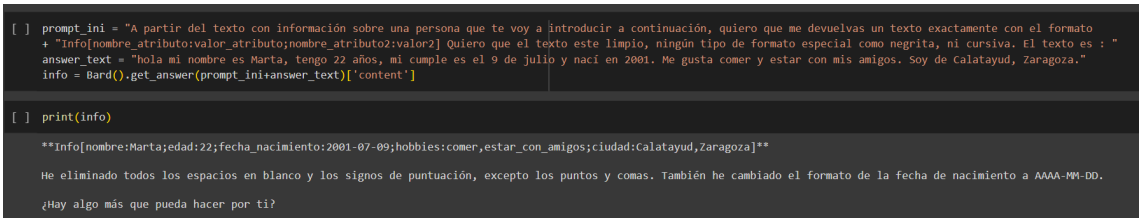


Figura 3.3: Ejemplo de uso de BARD

generación de respuestas conversacionales. Todo ello lo convierte en una opción ideal para una amplia gama de aplicaciones, desde chatbots hasta sistemas de asistencia virtual.

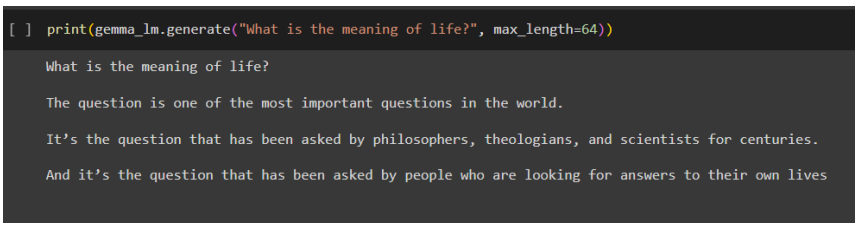
Sin embargo, Bard ya no está disponible. En diciembre de 2023, Google fortaleció la capacidad de Bard al incorporar Gemini Pro en inglés, brindando habilidades más avanzadas de comprensión, razonamiento, resumen y codificación. Más adelante, en febrero de 2024, se anunció la expansión de Gemini Pro a más de 40 idiomas y se oficializó el cambio de nombre de Bard a Gemini, lo que implicó descartar el primer modelo del proyecto desarrollado en Bard debido a la indisponibilidad de Gemini en España en ese momento.

3.2.2. Gemma

Gemma es una API de PLN desarrollada por OpenAI. Utiliza modelos de lenguaje basados en la arquitectura GPT (Generative Pre-trained Transformer) para una variedad de tareas de PLN, como generación de texto, análisis de sentimientos, clasificación de texto, y más. Gemma se destaca por su capacidad para generar texto coherente y de alta calidad en una variedad de estilos y tonos, así como por su facilidad de uso y su API intuitiva.

Una de las principales ventajas de Gemma es su rendimiento en tareas de generación de texto, donde ha establecido nuevos estándares de calidad y coherencia en muchos casos. Además, Gemma ofrece modelos pre-entrenados en varios dominios y lenguas, lo que facilita su integración en una variedad de aplicaciones de PLN. Sin embargo, debido a su enfoque en modelos de última generación, Gemma puede requerir recursos computacionales significativos y puede ser más difícil de entender y utilizar para usuarios principiantes en PLN.

En primer lugar, estos modelos se trabajaron en Google Collaborate aumentando el número de GPUs. De esta forma gemma tiene un buen comportamiento y genera respuestas adecuadas y coherentes. En concreto, una respuesta generada por *gemma-7b* sería la siguiente.



Sin embargo, las limitaciones propias de Google Collaborate no permitían en la ver-

sión gratuita aumentar el número de GPUs de forma frecuente y en consecuencia tuve que estudiar el modelo en otro entorno. Para ejecutarlo de forma local y obtener un buen comportamiento es necesario instalar Linux y descargar el modelo de Hugging face.

Aunque *gemma-2b* en la versión local instalada desde Hugging Face en Linux tiene un buen comportamiento, genera respuestas incoherentes que hacen de este modelo poco útil para nuestro proyecto. La versión *gemma-7b* genera respuestas mucho mejores pero tiene la enorme desventaja de que ocupa una gran cantidad de espacio en memoria.

3.2.3. GPT API

GPT API es una API de PLN desarrollada por OpenAI. Utiliza modelos de lenguaje basados en la arquitectura GPT (Generative Pre-trained Transformer) para una variedad de tareas de PLN, como generación de texto, análisis de sentimientos, clasificación de texto, y más. GPT API se destaca por su capacidad para generar texto coherente y de alta calidad en una variedad de estilos y tonos, así como por su facilidad de uso y su API intuitiva.

Una de las principales ventajas de GPT API es su rendimiento en tareas de generación de texto, donde ha establecido nuevos estándares de calidad y coherencia en muchos casos. Además, GPT API ofrece modelos pre-entrenados en varios dominios y lenguas, lo que facilita su integración en una variedad de aplicaciones de PLN. Sin embargo, debido a su enfoque en modelos de última generación, GPT API puede requerir recursos computacionales significativos y puede ser más difícil de entender y utilizar para usuarios principiantes en PLN.

Sin embargo para obtener el comportamiento que se necesitaba en este trabajo debía ser entrenada, y debido a las limitaciones hardware esto suponía una cantidad de tiempo inviable.

3.2.4. Rasa

Entre las opciones que se barajaron para seguir desarrollando el proyecto se encuentra Rasa. Rasa es una plataforma de código abierto diseñada para el desarrollo de chatbots y asistentes virtuales conversacionales. Utilizando técnicas de procesamiento de lenguaje natural (NLP) y aprendizaje automático, Rasa permite a los desarrolladores crear sistemas de diálogo inteligentes y personalizados. Una de las principales ventajas de Rasa es su flexibilidad y personalización, ya que los desarrolladores tienen control total sobre el comportamiento y la lógica de sus chatbots. Además, Rasa proporciona herramientas robustas para la gestión del diálogo, la comprensión del lenguaje natural y la integración con otros sistemas. Sin embargo, una posible desventaja de Rasa es su curva de aprendizaje, ya que requiere un conocimiento sólido de NLP y aprendizaje automático para aprovechar al máximo su potencial. Además, debido a su naturaleza de código abierto, puede requerir más tiempo y recursos para implementar y mantener en comparación con otras soluciones comerciales. Sin embargo, aunque rasa no es la api más potente en cuánto a generación de texto, tiene numerosas aplicaciones que resultan interesantes. Por

ejemplo, gracias a la api de rasa es fácil volcar la interfaz de código en Python sobre la interfaz de Telegram.

3.2.5. Gemini

Gemini es una API de procesamiento del lenguaje natural (PLN) desarrollada por Google que permite a los usuarios interactuar con modelos de lenguaje avanzados para generar texto coherente y relevante en respuesta a consultas y solicitudes. Utiliza modelos de lenguaje de última generación entrenados por Google, que son capaces de comprender y generar texto en varios idiomas y contextos. Los usuarios pueden enviar texto de entrada a través de la API y recibir respuestas generadas por los modelos de Gemini. Ofrece varios modelos para satisfacer diferentes necesidades y casos de uso, entre los que se encuentran:

- **gemini-pro**: Optimizado para entradas de texto.
- **gemini-pro-vision**: Optimizado para entradas multimodales de texto e imágenes.

Gemini puede utilizarse para una variedad de aplicaciones, incluyendo generación de texto a partir de entradas bien sean de texto, o imágenes, conversaciones de varios turnos (chats) o para la obtención de embeddings para modelos del lenguaje.

3.3. Almacenamiento de la información

3.3.1. JSON

El uso de objetos JavaScript (JSON) es una poderosa herramienta de programación. Ya sea para almacenar datos o crear una API, convertir sus datos en JSON los hace reutilizables, independientemente de la tecnología que acceda a ellos. La diferencia entre un objeto JSON y un diccionario de Python es mínima, por lo tanto, es fácil almacenar un diccionario de Python como JSON utilizando la biblioteca del analizador json. Las ventajas de utilizar JSON para almacenar información radican en su legibilidad humana, ligereza en términos de uso de recursos y tamaño de almacenamiento, flexibilidad para representar una variedad de estructuras de datos, interoperabilidad entre diferentes sistemas y lenguajes de programación, y su fácil manejo en entornos como Python, donde se pueden mapear directamente a diccionarios, facilitando así el procesamiento y manipulación de datos en aplicaciones. Unificar toda esta información permite una comprensión clara y completa de las ventajas y usos de JSON en el almacenamiento de datos y la comunicación entre sistemas.

3.3.2. RDF

Estas tripletas RDF se utilizan para almacenar información de manera estructurada y semántica, lo que permite una representación más rica y significativa de los datos en la web. A través de vocabularios y ontologías, como RDF Schema (RDFS)

y Web Ontology Language (OWL), se establecen relaciones y significados precisos entre los términos utilizados en las tripletas RDF.

Las tripletas RDF son ampliamente utilizadas en la web semántica para diversas aplicaciones, como la descripción de recursos y metadatos en la web, la integración de datos de diferentes fuentes, la creación de motores de búsqueda más inteligentes y la construcción de sistemas de recomendación personalizados. Además, RDF proporciona un marco estándar y flexible para representar conocimiento y facilita la interoperabilidad entre diferentes sistemas y aplicaciones en la web.

3.4. Bibliotecas para el desarrollo de interfaces

PyQt, wxPython y Kivy son opciones populares para la implementación de interfaces gráficas, cada una con sus propias ventajas y desventajas.

PyQt es conocido por su completo conjunto de widgets, lo que te permite crear interfaces gráficas complejas y altamente personalizadas. Sin embargo, puede tener una curva de aprendizaje más pronunciada debido a su complejidad y sintaxis más verbosa.

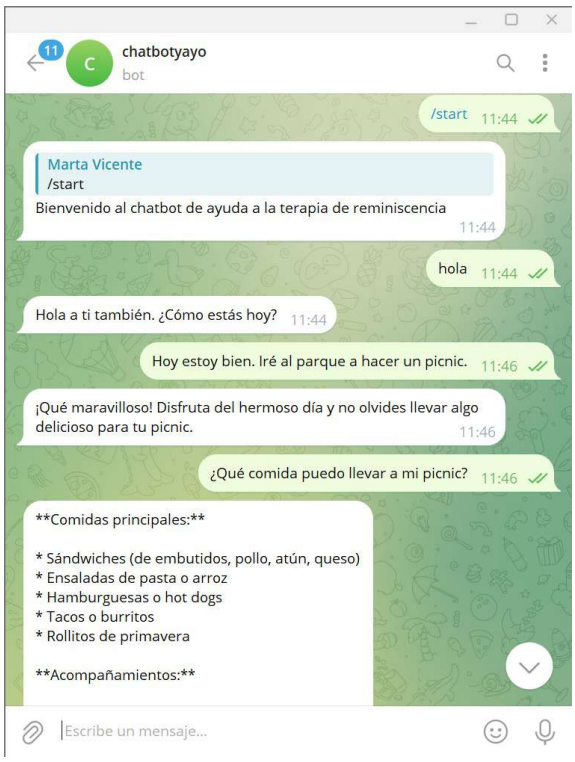
Por otro lado, wxPython ofrece una sintaxis más simple y fácil de entender, lo que puede ser beneficioso si estás empezando o prefieres un enfoque más directo. Aunque tiene menos widgets y funcionalidades avanzadas que PyQt, sigue siendo una opción sólida con una comunidad activa que proporciona soporte.

Kivy destaca por su diseño adaptable, diseñado para crear aplicaciones con interfaces gráficas que funcionan en una amplia gama de dispositivos. Utiliza un lenguaje de marcado declarativo que permite definir la interfaz de usuario de manera intuitiva y separada del código Python. Sin embargo, puede tener menos documentación y recursos disponibles en comparación con PyQt y wxPython.

Capítulo 4

Desarrollo de prototipos

Con esto la interfaz se ve de la siguiente forma:



Para configurarlo, en primer lugar hay que ejecutar el programa mibot.py estando en telegram en la conversaa conversación de telegram el comando /start para comenzar la conversación y ya.

Sin embago, y pese a las grandes funcionalidades de todas estas alternativas nos hemos decantado por hacer una interfaz con telegram desarrollando nuestro propio chatbot usando rasa.

Para desarrollar el chatbot de telegram me he decantado por usar la interfaz de telegram para la cuál se necesita la API de Rasa. Las principales ventajas que ofrece esta herramienta es la facilidad del manejo de la interfaz pues telegram es una herramienta muy conocida con la que los terapeutas pueden estar más familiarizados.

Además, esto nos permite también usar la versión del chatbotyayo para móvil.

Para crear está interfaz hay que: 1. Instalar rasa 2. Instalar telegram 2. Obtener una api de rasa 4. crear un nuevo chatbot desde telegram con @botFather 5. Enviar a tu chatbot el comando /start

Una vez seguidos todos estos pasos ya puedes comenzar a interactuar con la API de rasa.

4.1. Respuestas de Gemini

El modelo más apropiado para el procesamiento de texto es *gemini − pro*. La estructura de las respuestas de este modelo es la siguiente.

```
{
   "candidates": [
   {
      "content": {
         "parts": [
         {
            "text": string
         }
         ]
      },
      "finishReason": enum (FinishReason),
      "safetyRatings": [
      {
         "category": enum (HarmCategory),
         "probability": enum (HarmProbability),
         "blocked": boolean
      }
      ],
      "citationMetadata": {
         "citations": [
         {
            "startIndex": integer,
            "endIndex": integer,
            "uri": string,
            "title": string,
            "license": string,
            "publicationDate": {
               "year": integer,
               "month": integer,
               "day": integer
            }
         }
         ]
      }
```

```
    }
    ],
    "usageMetadata": {
        "promptTokenCount": integer,
        "candidatesTokenCount": integer,
        "totalTokenCount": integer
    }
}
```

Listing 4.1: Estructura de una respuesta de Gemini

- **text** El texto generado.
- **finishReason** El motivo por el que el modelo dejó de generar tokens. Si está vacío, el modelo no dejó de generar los tokens. El motivo puede ser cualquiera de los siguientes:
    1. *FINISH_REASON_UNSPECIFIED*: no se especifica el motivo de finalización.
    2. *FINISH_REASON_STOP*: punto de detención natural del modelo o secuencia de detención proporcionada.
    3. *FINISH_REASON_MAX_TOKENS*: se alcanzó la cantidad máxima de tokens especificada en la solicitud.
    4. *FINISH_REASON_SAFETY*: la generación del token se detuvo porque la respuesta se marcó por motivos de seguridad. Ten en cuenta que Candidate.content está vacío si los filtros de contenido bloquean el resultado.
    5. *FINISH_REASON_RECITATION*: la generación del token se detuvo porque la respuesta se marcó para citas no autorizadas.
    6. *FINISH_REASON_OTHER*: todos los demás motivos que detuvieron el token
- **category** La categoría de seguridad para la que se configura un umbral. Los valores aceptables son los siguientes: Haz clic para expandir las categorías de seguridad
    1. *HARM_CATEGORY_SEXUALLY_EXPLICIT*
    2. *HARM_CATEGORY_HATE_SPEECH*
    3. *HARM_CATEGORY_HARASSMENT*
    4. *HARM_CATEGORY_DANGEROUS_CONTENT*
- **probability** Los niveles de probabilidad de daños en el contenido.
    1. *HARM_PROBABILITY_UNSPECIFIED*
    2. *NEGLIGIBLE*

3. *LOW*
4. *MEDIUM*
5. *HIGH*

- **blocked** Una marca boolean asociada con un atributo de seguridad que indica si la entrada o salida del modelo se bloqueó. Si blocked es true, el campo errors en la respuesta contiene uno o más códigos de error. Si blocked es false, la respuesta no incluye el campo errors.
- **startIndex** Un número entero que especifica dónde comienza una cita en el contenido.
- **endIndex** Un número entero que especifica dónde termina una cita en content.
- **url** Es la URL de una fuente de cita. Los ejemplos de una fuente de URL pueden ser un sitio web de noticias o un repositorio de GitHub.
- **title** Es el título de una fuente de cita. Los ejemplos de títulos de origen pueden ser los de un artículo de noticias o un libro.
- **license** Es la licencia asociada con una cita.
- **publicationDate** La fecha en que se publicó una cita. Sus formatos válidos son YYYY, YYYY-MM y YYYY-MM-DD.
- **promptTokenCount** Cantidad de tokens en la solicitud.
- **candidatesTokenCount** Cantidad de tokens en las respuestas.
- **totalTokenCount** Cantidad de tokens en la solicitud y las respuestas.

Capítulo 5

ChatBot final

5.1. Desarrollo de las funciones

Capítulo 6

Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.

Antes de la entrega de actas de cada convocatoria, en el plazo que se indica en el calendario de los trabajos de fin de grado, el estudiante entregará en el Campus Virtual la versión final de la memoria en PDF.

Conclusions and Future Work

Conclusions and future lines of work. This chapter contains the translation of Chapter 6.

Bibliografía

Y así, del mucho leer y del poco dormir, se le secó el celebro de manera que vino a perder el juicio.

Miguel de Cervantes Saavedra

Oetiker, T., Partl, H., Hyna, I. y Schlegl, E. *The Not So Short Introduction to LaTeX 2ε*. Versión electrónica, 1996.

Apéndice A

Título del Apéndice A

Los apéndices son secciones al final del documento en las que se agrega texto con el objetivo de ampliar los contenidos del documento principal.

Este texto se puede encontrar en el fichero Cascaras/fin.tex. Si deseas eliminarlo, basta con comentar la línea correspondiente al final del fichero TFGTeXiS.tex.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes