Universidad Internacional de la Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

BOT conversacional para acompañar y cuidar a personas mayores

Trabajo Fin de Máster

presentado por: Diego de los Reyes Rodríguez

Dirigido por: Esteve Nadal Roig

Ciudad:Madrid

Fecha: 21 de julio de 2022

Índice de Contenidos

Índice de Ilustraciones

Índice de Tablas

Resumen

Las personas pierden progresivamente su autonomía a medida que envejecen. Esto hace necesario que cuenten con ayuda de terceros.

Se desarrolla un BOT para Telegram usando la plataforma Dialogflow en español para ayudar a los usuarios, acompañándoles y cuidándoles con acciones como recordarles su medicación o avisar a sus cuidadores de su estado real.

El objetivo principal es lograr que el BOT sea capaz de averiguar la intención del usuario en base al texto de entrada con una tasa de acierto superior al 85 %.

El BOT se les proporcionó a diez personas mayores con sus jóvenes cuidadores durante el desarrollo del producto. El modelo del lenguaje ajustado en Dialogflow con el conjunto de datos creado con sus interacciones con el BOT, alcanzó una tasa de acierto del 89.66 % en la tarea de detección de la intención. El grado de satisfacción final de los usuarios ha sido de 4.75 sobre 5.

Palabras Clave: BOT conversacional, cuidadores, personas mayores, procesamiento del lenguaje natural, modelo del lenguaje ajustado

Abstract

Most people progressively lose their autonomy as they get older. This makes it necessary for them to have third party help.

A BOT application for Telegram chat has been developed using the Dialogflow platform in Spanish to help users accompanying and caring them with actions such as reminding of their medication or alerting their caregivers of real conditions.

The main objective is to make the BOT application able to find out the user's intent based on the input text with a hit rate of over 85 %.

Ten elderly people and their young caregivers were provided with the BOT application during the product development. The language model fine-tuned at the Dialogflow platform with the dataset created with people's interactions with the BOT application achieved a hit rate of 89.66 % on the intent detection task. The final user satisfaction rating was 4.75 out of 5.

Keywords: conversational BOT, caregiver, elderly people, natural language processing, fine-tuned language model

Capítulo 1

Introducción

¿Es posible que la Inteligencia Artificial, mediante un BOT conversacional (desde ahora BOT), sea capaz de identificar las intenciones de las personas mayores que lo usen, aprendiendo en base a sus diálogos? ¿Estas personas mayores quieren que ese BOT forme parte de sus vidas para acompañarlos y cuidarlos, o por el contrario, les causa rechazo?

En el mundo en el que vivimos no todas las personas tienen las mismas oportunidades de llevar una vida feliz y saludable. Muchas personas, por el hecho de nacer en un determinado lugar del mundo, por su raza, sexo, edad, enfermedades o discapacidad, o por otros muchos factores, no pueden disfrutar de una vida plena.

Este trabajo se centra en ayudar a uno de estos grupos, concretamente, a las personas mayores, a sentirse más autónomas y menos dependientes, gracias a un producto que utiliza la Inteligencia Artificial.

El cese de las capacidades funcionales de los distintos órganos y sistemas del organismo humano y el declive de las capacidades cognitivas y de funciones motoras comienza, tal y como se explica en el libro "Cuando el cerebro envejece" (Bentivoglio & Zucconi, 2018), a medida que el ser humano se hace mayor.

Según en el libro "Neurodegeneración"(Gregorio, 2018), a día de hoy este deterioro es irreversible. Las enfermedades que conllevan a la muerte de neuronas no tienen todavía cura. Entender todos los procesos neuronales es un gran reto biomédico en el siglo XXI. El libro explica además que entre las enfermedades provocadas por este deterioro se encuentra el Alzheimer (EA). Para su diagnóstico se pueden establecer criterios que indican pérdidas de memoria, desorientación, depresión o dificultad para realizar actividades complejas de la vida diaria.

Este deterioro conlleva una pérdida de capacidades, que provocan consecuencias tanto emocionales como sociales en las personas afectadas. Se manifiestan en la aparición de emociones negativas, como la soledad (Doblas & Conde, 2018).

En (Gregorio, 2018) se explica que el problema del envejecimiento puede afectar, además de a áreas biológicas y psicológicas, a niveles sociales y políticos.

En el artículo (Carlos & Gualdrón, 2018), se expone el progresivo envejecimiento de las poblaciones europeas. En 2018, un 18,9 % de la Unión Europea tenía 65 años o más. La iniciativa Active and Assisted Living ("AAL Home 2020 - AAL Programme", s.f.), fundada en 2008, pronostica para 2070 que más de la mitad de la población de la Unión Europea será mayor de 65 años.

España, según (Conde-Ruiz, 2021), se encuentra en la actualidad en pleno proceso de envejecimiento. La tasa de dependencia, como se puede apreciar en 1.1, no ha parado de crecer en los últimos años. Según estadísticas del INE, AIReF y Eurostat, el aumento de población mayor va a continuar hasta el año 2050, año en el que puede llegar a multiplicarse prácticamente por dos, como se aprecia en la figura 1.2. Esto ocasionará que en 2050 España sea uno de los países más envejecidos.

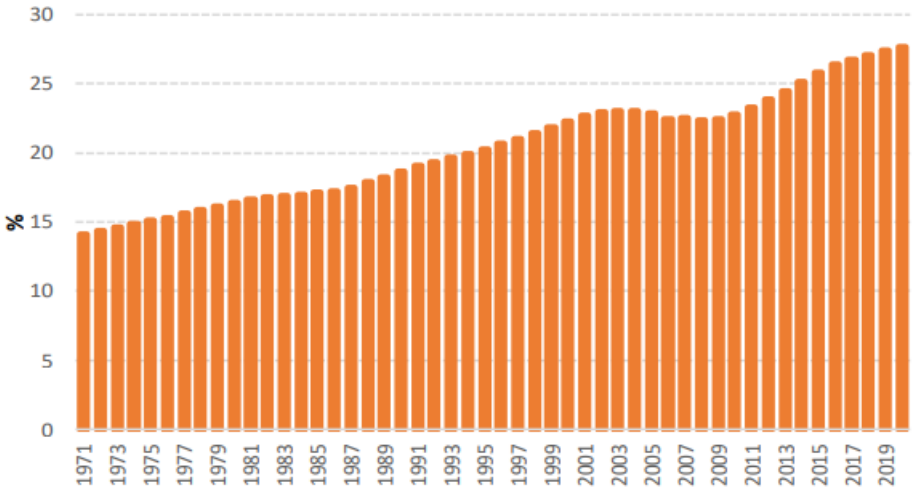


Figura 1.1: Tasa de dependencia a los 67 años (1971 a 2020) (Conde-Ruiz, 2021)

Este envejecimiento, según (Sánchez, 2014) conlleva un incremento de demanda de servicios socio-sanitarios que tienen como fin mantener la autonomía y calidad de vida de las personas mayores.

En el mercado se pueden encontrar diferentes soluciones empresariales que, de un modo u otro, pueden ayudar y acompañar a personas mayores, como pueden ser los asistentes

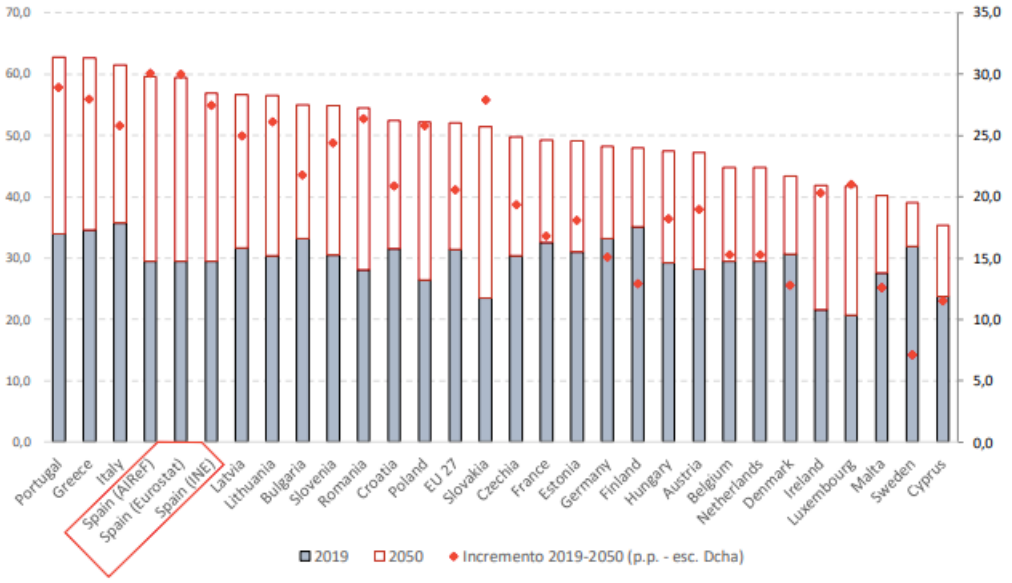


Figura 1.2: Tasa de dependencia a los 65 años. Año 2050 (Conde-Ruiz, 2021)

de Google y Alexa o robots asistentes.

La contribución de este trabajo se centra en ayudar a las personas mayores a sentirse más autónomas gracias a un BOT conversacional. A partir de este BOT, se elabora una base de datos de diálogos de los usuarios, que sirva para el entrenamiento de diferentes modelos dentro de un BOT con un diseño arquitectónico escalable, que permite utilizar diferentes modelos del lenguaje para diferentes tareas. Este BOT está dotado de iniciativa propia para iniciar una conversación con los usuarios, ya sea para ayudar o acompañar a una persona mayor o para informar a su cuidador de su estado.

1.1. Motivación

La tecnología se puede aplicar a múltiples tareas, entre las cuales, puede contribuir a hacer un mundo en el que se encuentre la equidad entre todas las personas, ya sea con robots, aplicaciones o prótesis.

Uno de los colectivos con necesidades especiales son las personas mayores, quienes se pueden llegar a sentir solas, sin nadie que les pueda cuidar y acompañar, mientras observan cómo van perdiendo facultades físicas y mentales. Tarde o temprano, cada uno de nosotros se verá afectado por el deterioro físico y cognitivo que se produce al envejecer.

Este trabajo se justifica por el hecho de que, a día de hoy, no existe una solución real capaz de acompañar y cuidar a cada persona mayor, cada vez hay más personas mayores

en el mundo, y no se ha encontrado una forma de revertir el proceso de envejecimiento. Por este motivo, se pretende dar un pequeño paso más para ayudar a personas con necesidades especiales mediante el uso de la tecnología. Para ello, se desarrolla un BOT conversacional dirigido a las personas mayores, que les pueda acompañar y hacerles sentir un poco más autónomos e independientes.

Este BOT no pretende ser en un sustituto de los familiares o cuidadores, sino un complemento de los mismos cuando éstos no puedan estar físicamente al lado de sus mayores.

Es, por tanto, una motivación social y un uso responsable de la tecnología la que incita al desarrollo de este trabajo.

1.2. Planteamiento del trabajo

Para ayudar a estas personas mayores a mejorar su día a día, se desarrolla un BOT conversacional, que implementa un sistema de diálogo para interactuar con estas personas. El trabajo se encuadra en el campo del Procesamiento del Lenguaje Natural (PLN). El PLN es un área de la Inteligencia Artificial que hace accesible el lenguaje humano a las máquinas, para lo cual, debe interpretar las oraciones proporcionadas por un humano durante un proceso de comunicación. El objetivo principal es lograr que el BOT conversacional sea capaz de averiguar la intención del usuario en base al texto de entrada con una tasa de acierto superior al 85 %.

La interfaz del BOT con el usuario consiste en un BOT de Telegram, para que los usuarios puedan llevarlo fácilmente en su dispositivo móvil sin necesidad de instalar y aprender a utilizar nuevas aplicaciones. El código se ha desarrollado en lenguaje Python y se despliega en la plataforma *Amazon Web Services* (AWS).

El BOT no sólo se comunica, tanto de forma proactiva como reactiva, con los mayores, sino que, además, puede enviar mensajes a sus contactos para informar del estado o necesidades de la persona que cuidan.

Los pasos que se dan en este trabajo para convertir la idea en un producto que puedan tener los usuarios finales son:

- Estudiar el contexto y estado del arte actual del PLN y los sistemas conversacionales y su aplicación a resolver necesidades de personas mayores.

- Definir un objetivo general y objetivos específicos, basados en la motivación y estado del arte.
- Escoger una metodología de trabajo adecuada para el desarrollo del producto.
- Describir los requisitos necesarios que permitan cumplir los objetivos planteados.
- Desarrollar el producto.
- Evaluar su calidad y utilidad.
- Determinar en las conclusiones el grado de éxito del trabajo en base a la evaluación.
- Establecer una serie de posibles trabajos futuros que pudieran partir del presente trabajo.

Durante todo el proceso de trabajo, se cuenta con un grupo de usuarios finales, formado por personas mayores y personas jóvenes, quienes, mediante interacciones con el BOT y encuestas, opinan sobre su utilidad y proponen mejoras.

1.3. Estructura de la memoria

En el primer capítulo de introducción, se presenta el trabajo, el público al que va dirigido y se justifica su utilidad. Se explica también cómo se plantea el trabajo para llevar a cabo el desarrollo del BOT conversacional y cómo se va a contar con un grupo de usuarios involucrados durante el ciclo de vida del proyecto.

A continuación, se presenta el contexto y estado del arte, que explica en qué punto está actualmente la investigación sobre la que se apoya este trabajo, profundizando en los actuales modelos del lenguaje con Redes Neuronales Profundas (Deep Neuronal Networks, DNN). Se explican las conclusiones obtenidas y, finalmente, la contribución del trabajo.

El tercer capítulo define los objetivos a cumplir con el BOT, la metodología a seguir y herramientas a utilizar para alcanzar los objetivos. Se describe cómo se va a evaluar si se han alcanzado los objetivos y, por tanto, el éxito del proyecto.

En el capítulo cuatro, se identifican los requisitos iniciales a satisfacer para cumplir los objetivos planteados. Estos requisitos se describen como historias de usuario, que contribuyen directamente a alcanzar el objetivo final, e historias técnicas, necesarias, de forma indirecta, para llegar al objetivo y alcanzar los objetivos específicos.

En el quinto capítulo se encuentra la descripción del software desarrollado, su diseño arquitectónico y cada una de las piezas que forman parte de él. Se explica su despliegue en servidores AWS y las pruebas realizadas para su correcto funcionamiento.

En el capítulo sexto, se evalúa el software desarrollado. Se miden la cobertura de código y número de pruebas automáticas pasadas, se exponen los resultados de las encuestas de satisfacción de uso del BOT de los usuarios y se muestran los resultados de la tarea de identificación de la intención en base al conjunto de datos creado.

En el apartado de conclusiones se revisan los objetivos a cumplir y se determina si se han alcanzado en base a los resultados. Se indican posibles trabajos futuros, obtenidos, principalmente, de las encuestas de los usuarios.

La bibliografía recoge referencias a explicaciones teóricas, investigaciones y trabajos previos de otros autores en los que el presente trabajo se ha apoyado, para que el lector pueda entenderlo y profundizar en los aspectos que necesite.

Finalmente, en los anexos, se adjunta el artículo científico que resume el presente trabajo, el cuaderno desarrollado para el análisis del conjunto de datos y evalulación del modelo y las encuestas enviadas a los usuarios para medir su grado de satisfacción.

Capítulo 2

Contexto y Estado del Arte

2.1. Procesamiento del lenguaje natural

En base a las diversas definiciones de PLN que se pueden encontrar en artículos como (Eisenstein, 2018) o (Vásquez y col., 2009), se puede definir el PLN como un área de la Inteligencia Artificial que hace accesible el lenguaje humano a las máquinas, para lo cual, debe interpretar las oraciones proporcionadas por un humano durante un proceso de comunicación.

El PLN resuelve diversas tareas, con diferentes técnicas que han evolucionado a lo largo de la historia, hasta llegar a generar modelos del lenguaje mediante DNN, en las que se centra este trabajo. Es necesario evaluar cómo de bien funciona el modelo para determinar cómo de bien desempeña cada tarea.

Entre las aplicaciones del PLN se encuentran los BOTs conversacionales. Un BOT conversacional, en sus diferentes componentes, utiliza modelos del lenguaje para resolver tareas de identificación de intenciones o generación de una respuesta al usuario en lenguaje natural.

Estos BOTs se invocan desde interfaces de usuario, como pueden ser las aplicaciones de mensajería tipo Telegram.

En este capítulo se explican las principales tareas que puede resolver el PLN, cómo se utilizan las DNN y los modelos más actuales que se usan para resolverlas, cómo se evalúa su desempeño y qué es y cuál es la arquitectura de un BOT conversacional.

2.1.1. Tareas del PLN

Gracias al PLN se pueden resolver multitud de tareas. Del libro de (Vajjala y col., 2020) y los artículos de (Vásquez y col., 2009), (Wolf y col., 2020), (Vicente y col., 2021) y (Eisenstein, 2018), se pueden extraer algunas de sus principales aplicaciones:

- **Clasificación de texto:** consiste en clasificar un texto de entrada en diferentes categorías en base a su contenido.
- **Extracción de información:** consiste en extraer información relevante del texto.
- **Recuperación de información:** consiste en encontrar dentro de una colección los documentos más relevantes, a partir de una consulta del usuario
- **Sistemas de respuesta a preguntas:** consiste en responder a preguntas planteadas en lenguaje natural.
- **Resumen de textos:** consiste en la creación de resúmenes cortos a partir de grandes cantidades de documentos, manteniendo lo esencial y conservando la esencia.
- **Traducción automática:** consiste en convertir una pieza de texto de un idioma de origen a uno de destino.
- **Modelado de temas:** consiste en descubrir la estructura temática de una gran cantidad de textos.
- **Generación de lenguaje natural:** consiste en producir texto en lenguaje natural.

2.1.2. Soluciones para PLN

Modelos del lenguaje con redes neuronales

Una forma de resolver tareas de PLN es utilizar modelos del lenguaje preentrenados basados en redes neuronales.

A lo largo de la historia, se han presentado diversas arquitecturas de DNN, desde las redes neuronales recurrentes hasta llegar a los Transformers. En 2017, un grupo de investigadores de Google presenta la arquitectura Transformers en el artículo **Attention Is All You Need** (Vaswani y col., 2017). En la figura 2.1 se muestra su arquitectura.

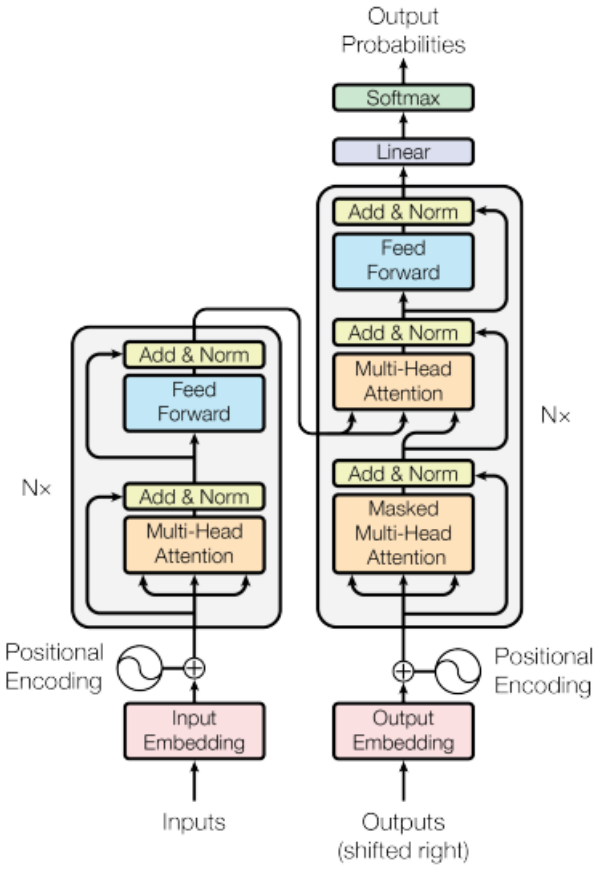


Figura 2.1: El Transformer - arquitectura. (Vaswani y col., 2017)

La aparición de esta nueva arquitectura dio lugar a nuevos modelos del lenguaje. En (Wolf y col., 2020) se puede ver el estado del arte de los últimos modelos, que se aprecian en la tabla 2.1, ordenados por su año de presentación.

Modelo	**Fecha**
BERT	2018
MarianMT	2018
RoBERTa	2019
GPT / GPT-2	2019
Trans-XL	2019
XLNet	2019
BART	2019
T5	2019
MMBT	2019
ALBERT	2019
DistilBERT	2019
XLM/RoBERTa	2019
GPT-3	2020
Reformer	2020
Longformer	2020
Electra	2020
MarIA	2022
Google PaLM	2022

Tabla 2.1: Modelos del lenguaje con DNN (Wolf y col., 2020)

En 2018 Google lanza el modelo BERT (Bidirectional Encoder Representations from Transformers). Este modelo se aplica en dos etapas, una inicial de pre-entrenamiento y otra de ajuste. El repositorio de BERT se encuentra en (Devlin, 2018).

Uno de los últimos modelos es GPT-3, de OpenAI, presentado en junio de 2020 en el artículo "Language Models are Few-Shot Learners"(Brown y col., 2020). Este modelo, basado en transformers, destaca por ser capaz de obtener muy buenos resultados para una tarea concreta realizando ajustes al modelo con observar muy pocos ejemplos de la tarea a resolver.

El artículo (Dale, 2021) define GPT-3 como un "narrador poco fiable", con mayor utilidad en generación de textos para casos como ficción o juegos de fantasía, pero de alto riesgo en casos en los que la respuesta deba ser verdadera.

En (Barbero & Vaca, 2022) se hace un repaso al estado del arte de los modelos del lenguaje en español. El primer modelo con éxito en el lenguaje español fue BETO, de la universidad de Chile, a finales de 2019.

Después de BETO aparecieron dos competidores muy fuertes. El primero, BERTIN (Rosa y col., 2022), del año 2022, basado en RoBERTa. Unos meses más tardes, MarIA (Gutiérrez-Fandiño y col., 2022), presentado en marzo de 2022 y financiado por el Plan Nacional de Tecnologías del Lenguaje. El modelo MarIA incluye una familia de modelos del lenguaje en español, con los más grandes y, posiblemente, los que mejor funcionan, como son RoBERTa-base, RoBERTa-large, GPT2 y GPT2-large, preentrenados con un corpus de 570GB de datos limpios, con 135 billones de palabras de la Biblioteca Nacional Española entre 2009 y 2019. El repositorio de código abierto se encuentra disponible en (Gutiérrez, 2022).

Plataformas PLN

Entre las plataformas que se pueden utilizar para el PLN, que se pueden emplear para el desarrollo de BOTs conversacionales, se destacan:

- **Google Dialogflow:** plataforma de Google para IA conversacional en lenguaje natural con agentes virtuales. Tal y como se indica su web (Google, s.f.), se utilizan los últimos modelos de comprensión del lenguaje natural basados en el ya citado BERT.
- **IBM Watson:** agente virtual inteligente, creado por IBM (IBM, s.f.). Permite crear chatbots sin necesidad de desarrollar código, aunque también es posible utilizar lenguajes de programación como Python o JavaScript. Estos chatbots se pueden integrar dentro de una web copiando y pegando un fragmento de código JavaScript en el HTML de un sitio web. Su uso es proporcionar soporte a los usuarios.
- **Amazon Lex:** plataforma creada por Amazon (Amazon, s.f.) para crear chatbots con IA conversacional que interactúen con el usuario por voz y por texto. Funciona en varios idiomas y se integra con otros servicios de AWS. Gracias a su infraestructura, permite al desarrollador centrarse en el diseño del chatbot, sin preocuparse del hardware ni del mantenimiento de ningún servidor.
- **Hugging Face**: plataforma (HuggingFace, s.f.) que ofrece una amplia colección de modelos pre-entrenados y conjuntos de datos de los que partir para realizar

fine-tuning un modelo. Se encuentran modelos del lenguaje en múltiples idiomas, ajustados para el desempeño de diferentes tareas. Gracias al buscador que se ofrece, se puede personalizar la búsqueda al idioma y tarea a resolver, para filtrar los modelos del lenguaje y seleccionar el que se necesite. Esta plataforma ofrece al usuario la posibilidad de almacenar los conjuntos de datos que creen y los modelos que hayan entrenado.

2.1.3. Evaluación

El proceso de evaluación de un modelo del lenguaje mide cómo de bien se comporta ante datos no vistos (Vajjala y col., 2020). Los resultados se pueden evaluar de forma intrínseca, esto es, comparando la salida del modelo con la salida etiquetada en el conjunto de datos, o extrínseca, evaluando la aplicación final.

Entre las métricas de evaluación intrínseca, se pueden utilizar la tasa de acierto, precisión, *recall* o *F1 score*. En tareas de clasificación, es común utilizar matrices de confusión.

La evaluación extrínseca mide cómo de buena es la aplicación final para resolver el problema objetivo, ya que, aunque se obtenga una buena métrica en la evaluación intrínseca, la aplicación podría no estar dando solución al problema real. Por esto, la evaluación extrínseca se realiza empleando métricas basadas en las reglas del negocio.

Los modelos del lenguaje, según (Eisenstein, 2018), idealmente se deberían evaluar de forma extrínseca, pero, dada su dificultad, normalmente se evalúan intrínsecamente, dado su menor coste, y, es de esperar, que ante una mejor evaluación intrínseca, la extrínseca aumente.

2.2. BOTs conversacionales

En (Vajjala y col., 2020), se define un BOT conversacional como un sistema interactivo que permite a los usuarios interactuar con él mediante lenguaje natural.

El origen de los BOTs conversacionales (Romero y col., 2020) se remonta a 1966, con el programa ELIZA, primer chatbot de la historia, con la capacidad de responder a una serie de preguntas, y de incitar al usuario a hablar cuando no le entiende con preguntas como *"¿por qué dices eso?"*, transmitiendo así la sensación de entender al usuario. En 1995, inspirado en ELIZA, se desarrolla Alicebot. En la historia reciente, podemos encontrar

a *Google Assistant* (Google), *Siri* (Apple) o *Cortana* (Microsoft), capaces de interactuar con el usuario mediante voz o texto y ejecutar diferentes acciones, como activar música u organizar citas médicas.

La arquitectura (Vajjala y col., 2020) de un BOT conversacional (ver en la figura 2.2) está formada por los siguientes componentes:

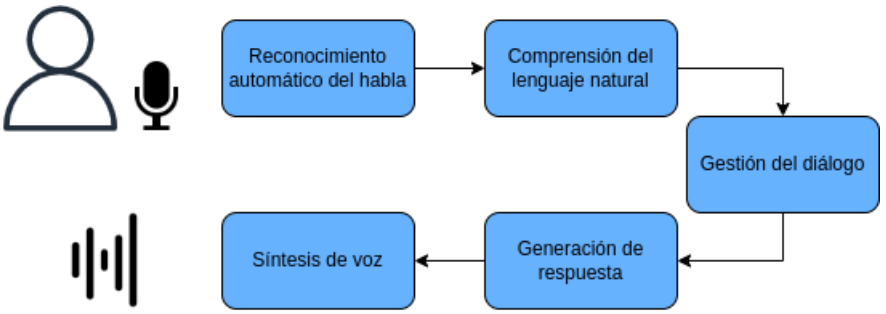


Figura 2.2: Flujo de agentes conversacionales (adaptado de Vajjala y col., 2020)

- **Reconocimiento automático del habla:** transforma las señales de audio del lenguaje hablado en lenguaje escrito.

- **Comprensión del lenguaje natural:** identifica la intención del usuario y extrae la semántica a partir del texto recibido del componente de reconocimiento automático del habla.

- **Gestión del diálogo:** a partir del análisis de la representación semántica, determina cómo debe continuar la conversación. Como salida, devuelve el concepto que se le quiere transmitir al usuario como respuesta en su siguiente turno de palabra. Se encarga de mantener el estado y flujo de la conversación, pidiendo más información al usuario en caso de necesitarla para completar la representación.

- **Generación de respuesta:** elige los conceptos que quiere transmitir al usuario y planifica cómo expresarlos.

- **Síntesis de voz:** convierte el texto a habla. Debe tener en cuenta el acento y la entonación de las palabras y sintetizar una onda de sonido.

2.3. Estado del arte

Una vez se ha explicado el contexto en el que se encuadra el problema a resolver y el marco teórico del PLN y agentes conversacionales que se aplica a la solución, se investigan trabajos académicos que se puedan tomar como referencia, a fin de determinar qué existe y qué se puede aportar en este trabajo.

Como consecuencia de los problemas derivados del envejecimiento, emergen soluciones tecnológicas, especialmente en forma de robots de servicio, para la asistencia a estas personas mayores, e iniciativas, como AAL ("AAL Home 2020 - AAL Programme", s.f.), un programa europeo que financia la innovación, el desarrollo de productos y servicios destinados a personas mayores que sufren los desafíos del envejecimiento y a sus cuidadores si necesitan ayuda.

En la tesis de (Sánchez, 2014) se realiza una ontología para modelar servicios AAL. La finalidad es ofrecer soporte a la movilidad y a la independencia de las personas mayores con deterioro cognitivo que sufren episodios de desorientación espacial.

El trabajo de (Carlos & Gualdrón, 2018) centra su atención en robots para el cuidado de personas mayores (geronrobótica). Estos robots se clasifican en:

- Médicos, para diagnosticar y llevar una vida saludable.
- De servicio, para ayudar en tareas domésticas.
- Sociales, para acompañar.
- Recreativos, para entretener.
- Educativos, para estimular el aprendizaje.
- Rehabilitadores, para ayudar en las limitaciones funcionales de los usuarios.
- Con potencial terapéutico, para ayudar ante problemas psicológicos negativos.

La mayoría de ellos, como puede verse en la figura 2.3, pertenecen a la categoría de robots de servicio.

El robot Maggie (ver figura 2.4), desarrollado por *Robotics Lab*, de la Universidad Carlos III, es un ejemplo de estos robots que atienden, vigilan y entretienen a los pacientes.

El trabajo de (los Santos Cicutto, 2017) realiza un estudio sobre la experiencia de uso de asistentes de voz sin GUI en personas mayores de 60 años con pocos conocimientos

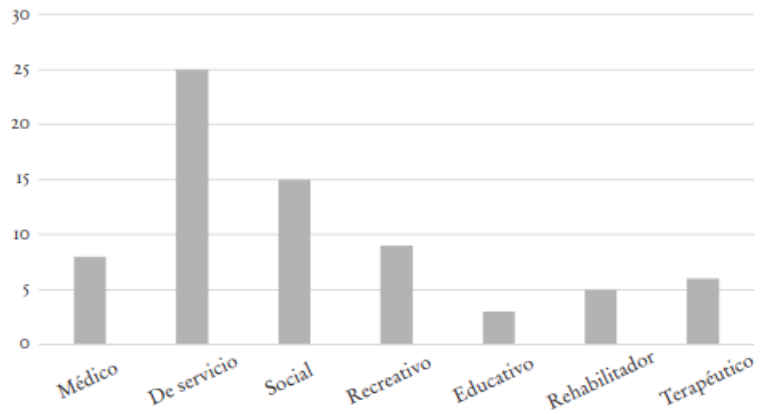


Figura 2.3: Tipos de robots encontrados (Carlos & Gualdrón, 2018)

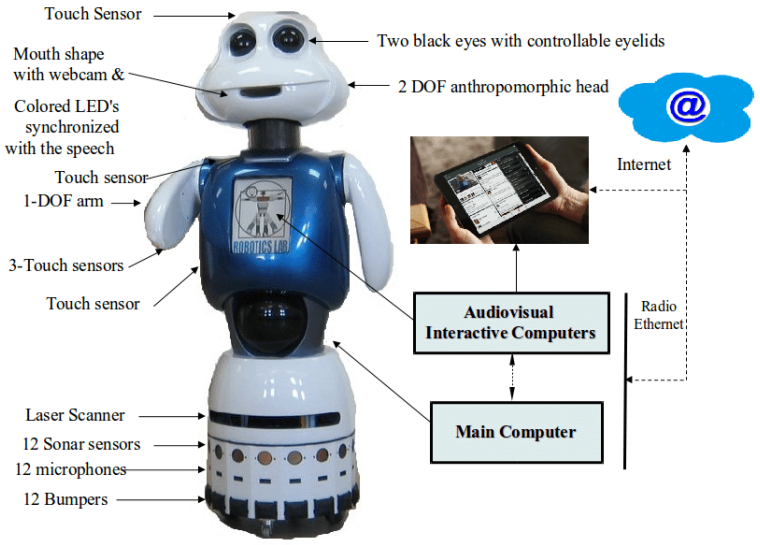


Figura 2.4: Robot Maggie (Alonso Martín y col., 2015)

informáticos, pero con algún mínimo contacto con las nuevas tecnologías. Este estudio concluye que, en base a las entrevistas realizadas, el uso de asistentes de voz cubre parte de sus necesidades, y el uso de estos asistentes puede romper las barreras, mejorando la comunicación de las personas mayores con estos asistentes y mejorar su relación con los ordenadores, al resultarles más sencillo e intuitivo su uso.

En el estudio de (Pradhan y col., 2019), se instalan altavoces inteligentes en los hogares de personas mayores durante tres semanas, para estudiar la interacción de estas personas con los altavoces. A estas personas se les hacen entrevistas diarias. El estudio concluye remarcando el potencial de esta tecnología para apoyar las interacciones sociales, ya que estas personas perciben cierto grado de acompañamiento del asistente. Finalmente, se

manifiesta la necesitad de explorar más a fondo el uso de asistentes de voz conversacionales para aliviar la soledad de diferentes poblaciones.

El trabajo de fin de grado de (Rodríguez, 2020), consistente en un agente conversacional para ayudar a las personas mayores a combatir la soledad, con la herramienta *Dialogflow* de *Google*, con lo que las conversaciones siempre son reactivas (iniciadas por el usuario). Este agente ofrece diversas funcionalidades, como adivinanzas, chistes, juegos o pedir cita al médico. Las funcionalidades de juegos y citas al médico están limitadas a que el usuario tenga una pantalla, es decir, no se pueden realizar completamente por voz, mientras que las de adivinanzas o chistes sí. En este trabajo, se realizan encuestas a los usuarios para medir la calidad del agente, que, por lo general, recibió buenas opiniones.

El estudio de (Kim, 2021) se centra en cómo las personas mayores perciben el beneficio de los altavoces. En sus resultados, se explica que la respuesta general a un asistente de voz fue positiva, gracias a la simplicidad de una interacción basada en el habla, y se valoró positivamente una respuesta cortés para completar la interacción con el asistente de voz, como expresar gratitud o dar retroalimentación sobre la calidad de las respuestas. Concluye con la necesidad de un buen diseño conversacional para interactuar con los mayores.

2.4. Conclusiones

En la actualidad, se pueden encontrar en el mercado robots asistenciales y diferentes trabajos e investigaciones sobre agentes conversacionales para ayudar y acompañar a las personas mayores, como el mencionado en la descripción (Rodríguez, 2020) con *Dialogflow*. Sin embargo, el problema sigue vigente a día de hoy, puesto que no se ha encontrado una herramienta tipo BOT destinada al cuidado de personas mayores en el mercado, validada y de uso extendido, que cuide y acompañe a nuestros mayores, y la biomedicina aún no es capaz de revertir el proceso degenerativo que se produce al envejecer.

La clasificación de robots de (Carlos & Gualdrón, 2018) aporta a este trabajo los diferentes dominios e intenciones en base a los tipos de robots encontrados, esto es, dominio médico, de servicio, social, recreativo, educativo y con potencial terapéutico, quedando fuera el tipo de robot rehabilitador, al requerir un robot físico.

Respecto a las soluciones para PLN, cabe destacar la variedad de alternativas que se pueden probar en la fase de desarrollo para resolver cada una de las tareas, desde modelos que funcionen en español, como GPT-3 (de pago), el recién aparecido modelo MarIA, hasta

plataformas como Dialogflow.

Por último, se aprecia que la aplicación de DNN a tareas de PLN es realmente joven y está creciendo muy rápidamente, por lo que es previsible que próximamente aparezcan nuevos modelos mayores y con mejores métricas.

2.5. Contribución

El presente trabajo aporta las siguientes contribuciones:

- Elaboración de un modelo capaz de detectar la intención del usuario, entrenado con una base de datos de diálogos de personas mayores y de personas jóvenes cuidadoras.
- Diseño arquitectónico de un BOT capaz de utilizar varios modelos de lenguaje para diferentes tareas de una forma escalable, para poder cambiar fácilmente de modelo.
- Desarrollo de un BOT con iniciativa propia para hablar a los usuarios sin que éstos inicien una conversación.
- Desarrollo de un BOT capaz de conectar a personas mayores y a sus cuidadores e informar a los segundos del estado de los primeros.

Capítulo 3

Objetivos y metodología de trabajo

En el presente capítulo, se describen los objetivos a solventar mediante el empleo de Inteligencia Artificial, el marco de trabajo Scrum y la colaboración de un grupo de usuarios mayores y jóvenes, que validan la utilidad del producto.

3.1. Objetivo general

El objetivo principal del trabajo consiste desarrollar un BOT conversacional que sea capaz de averiguar la intención del usuario en base al texto de entrada con una tasa de acierto superior al 85 %.

3.2. Objetivos específicos

Adicionalmente y vinculado al objetivo general, se busca alcanzar una serie de objetivos específicos, que se muestran en la figura 3.1.

- Elaborar un conjunto de datos que permita entrenar un modelo con una tasa de acierto superior al 85 % en la tarea de detección de la intención.
- Descubrir necesidades reales de las personas mayores a partir de la base de datos de diálogos y clasificarlas en intenciones según el diálogo.
- Investigar los diferentes modelos preentrenados y plataformas de PLN que se pueden usar para entrenar al BOT conversacional.

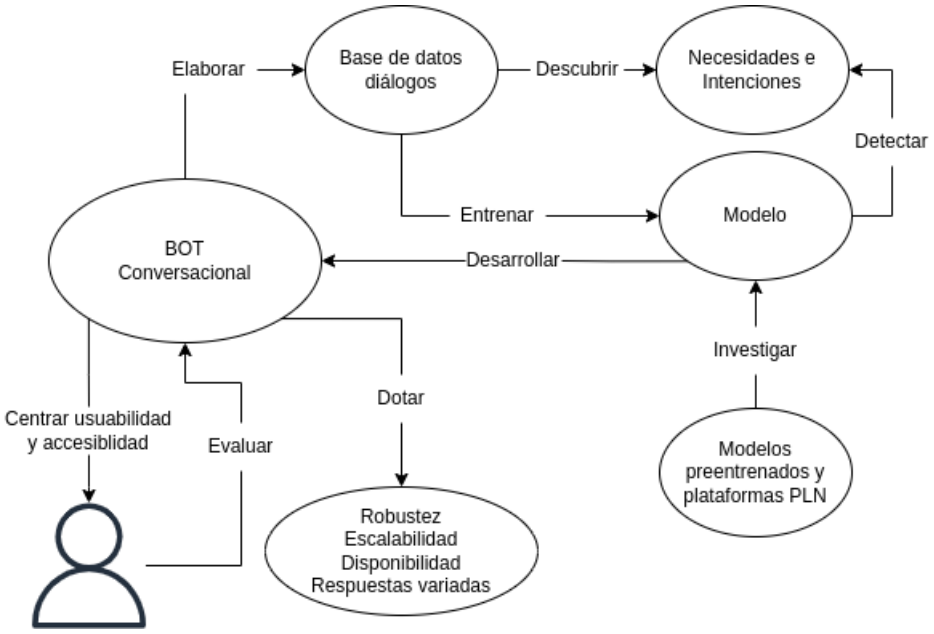


Figura 3.1: Objetivos Específicos

- Desarrollar un BOT conversacional empleando las bibliotecas de terceros que sean necesarias, que utilice el modelo entrenado y actúe en consecuencia de la intención identificada.
- Diseñar la arquitectura de un BOT conversacional que almacene información de usuarios y sea capaz de hablar de forma proactiva ante determinados eventos, de forma escalable, que permita al software crecer mediante la inclusión de nuevos componentes, según aumente el número de funcionalidades a implementar en base a las necesidades de mayores y jóvenes. La arquitectura debe estar preparada para el uso futuro del BOT conversacional desde plataformas diferentes a Telegram.
- Diseñar el servicio con previsión de que pueda estar disponible 24h al día durante 7 días a la semana.
- Evaluar la utilidad del BOT para las personas mayores y sus acompañantes, con una valoración global de satisfacción superior a 4.5 sobre 5.
- Dotar al sistema de robustez para recuperarse de errores sin que el usuario final se vea afectado.
- Centrar la usabilidad y accesibilidad hacia un público objetivo de personas mayores para que sean capaces de usarlo de forma autónoma.

- Crear un sistema de respuestas en lenguaje natural lo suficientemente variado para que el usuario sienta que tiene detrás una entidad no robótica.

3.3. Metodología de trabajo

Se escogen algunos aspectos del marco de trabajo Scrum (ver figura 3.2) para el desarrollo del producto, y se utiliza un tablero Kanban para la gestión de tareas.

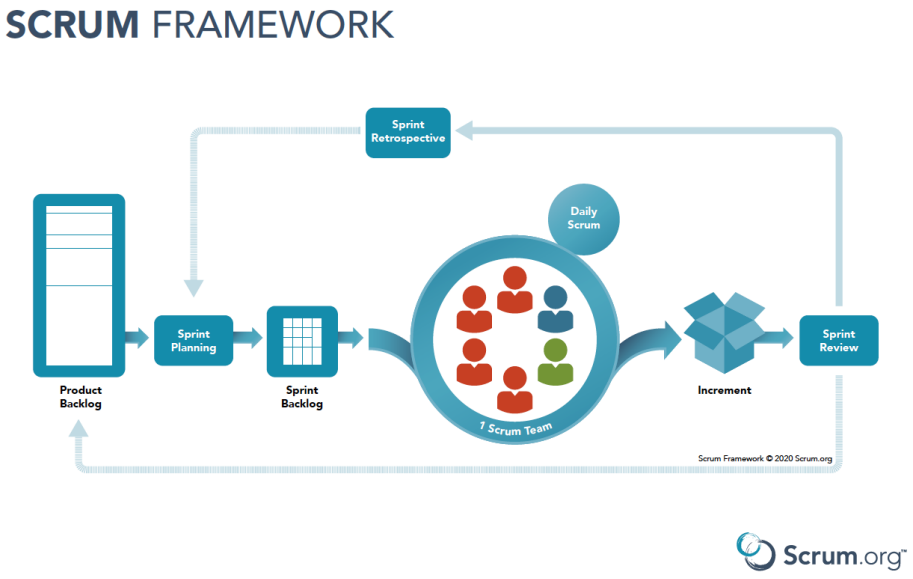


Figura 3.2: Scrum framework (https://www.scrum.org/)

El motivo principal de la elección de Scrum es poder realizar pequeños incrementos de valor cada poco tiempo a fin de contar con las opiniones de los usuarios finales lo más rápido posible, para adaptarse a sus necesidades y no invertir tiempo de desarrollar funcionalidades que sean poco o nada útiles. Se describen sus reglas, recogidas la Guía de Scrum de 2020 (Schwaber & Sutherland, 2020).

3.3.1. Roles de Scrum

Roles principales

Se distinguen tres roles dentro del equipo Scrum:

- **Product Owner:** Dueño de producto, que tiene contacto directo con el cliente, y es capaz de captar y transmitir sus necesidades al equipo de trabajo. Tiene poder para tomar decisiones sobre el producto.

- **Scrum Master:** Encargado de formar al equipo de trabajo y a los clientes finales sobre el marco de trabajo Scrum.
- **Equipo de desarrollo:** Equipo multidisciplinar en el que sus integrantes cuentan con todas las competencias necesarias para crear el producto.

Roles secundarios

Son los clientes a los que va dirigido el producto. En concreto, son las personas mayores y las personas jóvenes que participan como usuarios.

Se invita a participar en el proyecto a 16 personas mayores, junto con personas jóvenes que son familiares o conocidos suyos. De las 16 personas mayores, aceptan participar 10.

Se busca tener un grupo pequeño de usuarios y trabajar con sus diálogos en lugar de buscar bases de datos de diálogos de personas mayores, con el fin de tener contacto directo con ellos y captar sus necesidades de primera mano. Además de que, la creación de la base de datos de diálogo, es uno de los objetivos específicos.

3.3.2. Eventos de Scrum

El marco de trabajo Scrum contiene una serie de eventos, englobados dentro de un evento llamado *Sprint*.

- **El Sprint:** ciclo de un tiempo inferior a un mes, con un objetivo definido, durante el cual se aporta al producto un incremento de valor tangible por los clientes. En este trabajo la duración del sprint es de dos semanas.
- **Sprint Planning:** primer evento de un sprint en el que se define el objetivo del mismo y se establece el trabajo necesario para cumplirlo.
- **Daily Sprint:** reunión diaria donde se inspecciona el progreso hacia el objetivo del Sprint y se adapta el Sprint Backlog según sea necesario. Dado que en este trabajo sólo hay un integrante en el *Scrum Team*, este evento se convierte en una simple revisión del tablero para asegurarse de no perder de vista que se está trabajando en una tarea que acerque al objetivo del *Sprint*.
- **Sprint Review:** al final del sprint, se evalúa el incremento de valor aportado junto con usuarios finales, a quienes se les puede hacer una demostración del incremento. En este evento se reciben sus impresiones, para adaptarse a sus necesidades y descubrir

nuevas funcionalidades. Se valora su inclusión en próximos sprints si contribuyen al objetivo final del trabajo y no afectan en el tiempo hasta su entrega, o se incluyen como posible trabajo futuro en caso contrario.

Es importante que estén involucrados en el proceso de desarrollo para validar que el producto final cumpla sus expectativas y no identificar problemas básicos de diseño en fases tempranas de su desarrollo.

- **Sprint Retrospective:** como último evento de un sprint, la retrospectiva permite al equipo Scrum evaluar cómo ha funcionado el sprint y planificar la forma de mejorar el proceso para aumentar su calidad y efectividad.

3.3.3. Artefactos de Scrum

El marco de trabajo Scrum contiene los siguientes artefactos:

- **Product Backlog:** listado priorizado de aquello que sea necesario para mejorar el producto. El compromiso de este artefacto es el **Objetivo del Producto**, que es un objetivo a largo plazo del *Scrum Team*.
- **Sprint Backlog:** conjunto de elementos del *Product Backlog* seleccionados para alcanzar el **Objetivo del Sprint**.
- **Increment**: un paso que aporta valor al usuario final. Para que se considere un incremento, debe estar terminado, según el compromiso que indique la **Definición de Terminado**.

3.3.4. Tablero Kanban

Para la gestión de tareas, se emplea un tablero Kanban (ver figura 3.3), haciendo uso de la herramienta Trello.

En este tablero, se crean cinco columnas:

- **Product Backlog:** contiene todas las historias necesarias para la consecución del Product Goal.
- **Sprint Backlog:** contiene todas las historias necesarias, procedentes del Product Backlog, para la consecución del Sprint Goal del Sprint en curso.

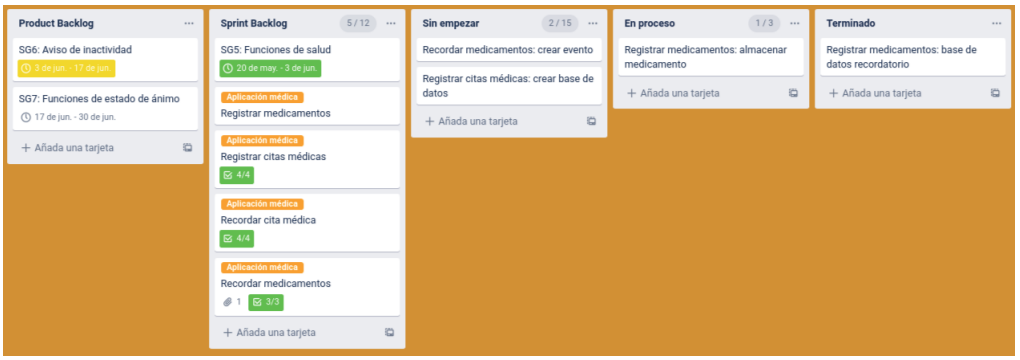


Figura 3.3: Tablero Kanban en Trello

- **Sin empezar:** contiene todas las tareas en las que se han desgranado las historias del Sprint en curso.
- **En proceso:** contiene todas las tareas, procedentes de la columna "Sin empezar", que se están realizando.
- **Terminado:** contiene todas las historias y tareas terminadas.

3.4. Planificación

Dado que el proyecto está acotado temporalmente, se establecen fechas clave de inicio y fin de sprint (ver tabla 3.1), fijando la duración de cada sprint en dos semanas. Se determina un Product Goal alineado con el objetivo general del presente trabajo:

BOT para acompañar y ayudar a personas mayores que salude, recuerde medicamentos y citas médicas, determine el estado de ánimo e informe a cuidadores.

Se establecen diferentes Sprint Goals que nos acerquen a ese Product Goal final.

La planificación es orientativa, por lo que puede variar durante el desarrollo. El motivo de una planificación adaptativa viene dado por el marco de trabajo Scrum, según el cual planifica un sprint nada más terminar el anterior, recogiendo las impresiones de los usuarios y teniéndolas en cuenta para fijar próximos objetivos, adaptándose a sus necesidades.

A continuación, se describe el objetivo de cada sprint.

3.4.1. Prototipo inicial

A modo de prueba de concepto, se desarrolla un primer prototipo rápido de un BOT para Telegram. Se despliega en un entorno local en el equipo de desarrollo y se hace uso de

ID	Sprint Goal	Fecha inicio	Fecha fin
1	Prototipo inicial	01/04/2022	07/04/2022
2	Envío del BOT al grupo de usuarios	07/04/2022	22/04/2022
3	Elaboración del conjunto de datos	22/04/2022	06/05/2022
4	Entrenamiento del modelo	06/05/2022	20/05/2022
5	Funciones de salud	20/05/2022	03/06/2022
6	Aviso de inactividad	03/06/2022	17/06/2022
7	Funciones de estado de ánimo	17/06/2022	30/06/2022

Tabla 3.1: Planificación temporal de cada sprint

GPT-3 para la generación de respuestas. Se da a probar a un grupo reducido de usuarios finales.

Este prototipo contempla las intenciones básicas de saludo y despedida y la intención por defecto que se dispara cuando no se entiende la intención final del usuario.

3.4.2. Envío del BOT al grupo de usuarios

Se envía una encuesta inicial a los usuarios, donde puedan expresar sus expectativas iniciales. Se desarrolla una primera versión no funcional del BOT capaz de registrar los diálogos de los usuarios. En esta versión el BOT no realiza ninguna tarea de PLN y se limita a almacenar la oración introducida por el usuario en una base de datos. El objetivo es familiarizar a los usuarios con el uso de un BOT de Telegram y crear una base de datos de diálogos para posteriores entrenamientos.

3.4.3. Elaboración del conjunto de datos

En base a los diálogos almacenados en la base de datos, se elabora un conjunto de datos para entrenar al modelo, cuya entrada es el texto introducido por el usuario y la salida la intención del usuario. La salida se etiqueta manualmente. Gracias a este conjunto de datos, se consigue un listado de necesidades de las personas mayores, que se corresponden con las intenciones etiquetadas. Durante esta fase, dado que se tiene contacto directo con los usuarios finales, es posible que se requiera su ayuda para terminar de identificar su intención en alguna oración concreta.

3.4.4. Entrenamiento del modelo

A partir del conjunto de datos obtenido, se entrenan varios modelos con diferentes modelos del lenguaje vistos en el estado del arte. El objetivo de esta fase es obtener una comparativa de la tasa de acierto de estos modelos en la tarea de detección de la intención del usuario.

3.4.5. Funciones de salud

Dentro del dominio de salud, se encuentran las intenciones de registrar medicación y citas médicas y recordarle al usuario cuándo debe tomar su medicación, además de avisarle de sus citas médicas.

En esta fase, se desarrollan los componentes software encargados de registrar en base de datos las peticiones del usuario y avisarle cuando sea necesario de su toma de medicación o cita médica.

3.4.6. Aviso de inactividad

Esta funcionalidad informa a los cuidadores si, durante un tiempo, la persona mayor de la que cuidan, no interactúa con el BOT.

3.4.7. Funciones de estado de ánimo

Se registra el estado de ánimo en función de cómo indique el usuario que se encuentra o cómo detecte el BOT su estado de ánimo. Se informa a los cuidadores si el estado registrado es negativo.

3.4.8. Finalización del proyecto

Se envía una encuesta final a los usuarios, donde puedan contrastar sus expectativas iniciales con el BOT final.

3.5. Evaluación

Para evaluar el cumplimiento de los objetivos, se llevan a cabo las siguientes acciones:

- Evaluación intrínseca del modelo del lenguaje entrenado para detectar la intención del usuario. Se mide la tasa de acierto (*accuracy*) que es capaz de alcanzar el BOT

en la detección de la intención. Tal y como ha sido definido en el objetivo principal, se considera que la tasa de acierto debe ser superior al 85 %.

- Se verifica que el conjunto de datos es adecuado si ha permitido entrenar el modelo con la tasa de acierto definida.
- Se comprueba la escalabilidad de la arquitectura, verificando que la inclusión de nuevos modelos del lenguaje, nuevas intenciones, nuevas acciones y nuevas interfaces, requieran la inclusión de nuevos componentes y no la modificación de componentes ya existentes.
- Se evalúa la calidad del software desarrollado mediante pruebas automáticas, con los criterios de cobertura de código, que debe ser superior al 90 %, y porcentaje de tests superados, que debe ser del 100 %. Estos tests deben asegurar el correcto funcionamiento del software en casos de uso correctos (es decir, el usuario introduce bien los datos) y en casos de uso alternativos (cuando el usuario realiza acciones incorrectas o existen fallos en la comunicación con componentes terceros).
- La disponibilidad del servicio no se evalúa, debido a que se hace uso de un servicio que garantiza dicha disponibilidad.
- La usabilidad, accesibilidad, utilidad para personas mayores, variedad de respuestas y grado de satisfacción de los usuarios, se miden mediante una encuesta final. Se comparan las expectativas de la encuesta inicial con las de la encuesta final, y, para que el objetivo se considere cumplido, la media de satisfacción global de la encuesta final debe ser superior 4 sobre 5.

En el capítulo de Evaluación (Capítulo 6), se evalúan los objetivos y se determina su cumplimiento en base a las métricas definidas en este apartado.

3.6. Stack tecnológico

En la figura 3.4 se muestra una imagen del stack tecnológico utilizado en el proyecto.

- **Telegram:** como interfaz de usuario se elige la aplicación de Telegram, para la cual se desarrolla un BOT. Telegram se puede instalar fácilmente en el móvil y permite crear bots de forma gratuita a los desarrolladores.

Figura 3.4: Stack tecnológico

- **Python:** se emplea el lenguaje de programación Python, dado que la mayor parte de bibliotecas terceras para inteligencia artificial que se utilizan en el desarrollo están desarrolladas en Python.

- **Docker:** el entorno local de ejecución se despliega en un contenedor *Docker*. En el mismo contenedor se ejecutan las pruebas automáticas con *pytest*. Gracias al uso de *Docker*, disponemos de un contenedor limpio con el software estrictamente necesario en sus versiones más adecuadas, sin tener que instalar en el equipo software adicional y permitiendo levantar y tirar los contenedores según necesidad.

- **Jupyter Notebook:** empleado para la creación del conjunto de datos y evaluación de los diferentes modelos del lenguaje en su tarea de detección de la intención del usuario. Gracias al uso de Jupyter Notebook se visualiza en una interfaz web de forma sencilla la ejecución del código Python.

- **AWS y Serverless:** el código desarrollado se aloja en AWS en funciones lambda, con la herramienta *serverless*. Proporciona un API Gateway como punto de comunicación con el BOT de Telegram. Se ha escogido AWS frente a otros servidores en la nube, como **Google Cloud** o **Microsoft Azure**, dado que, en base a experiencia previa en el entorno AWS, se ha podido verificar la viabilidad de su despliegue. El conjunto de datos se almacena, dentro de AWS, en una base de datos *DynamoDB*.

- **Dialogflow:** plataforma de Google donde se entrena el modelo en base al conjunto de datos de diálogos etiquetados.

Capítulo 4

Identificación de Requisitos

Los requisitos de usuario han sido elicitados del estado del arte y de la encuesta inicial a usuarios. Se identifican, en primer lugar, los usuarios a los que va dirigido el producto, para, a continuación, presentar las historias de usuario, que definen aquello que les aporta valor, y las historias técnicas, necesarias para sostener técnicamente las historias de usuario.

4.1. Historias de usuario

Con el fin de tener al usuario en el centro del producto, los requisitos funcionales se redactan como historias de usuario. Los usuarios que interactúan con el producto son:

- **Persona mayor:** Persona que tiene una edad superior a 65 años o está cerca de esa edad. Su rol es el de usuario principal, esto es, utiliza la aplicación para que le ayude y acompañe.
- **Cuidador:** Persona que no ha alcanzado la tercera edad, cercana a una persona mayor, con la que tiene contacto. Su rol es supervisar el estado de la persona mayor.

Las historias de usuario se agrupan en diferentes épicas:

- Épica 1: Conversación básica
- Épica 2: Registro del estado de ánimo
- Épica 3: Registro de medicinas
- Épica 4: Citas médicas
- Épica 5: Detección de inactividad

4.1.1. Épica 1: Conversación básica

HUCB001: Como usuario, quiero un BOT al que poder contarle mis necesidades para que me ayude y me acompañe.

HUCB002: Como usuario, quiero que el BOT me salude cuando yo le salude para tener un punto de inicio de conversación.

HUCB003: Como usuario, quiero que el BOT se despida cuando yo me despida para tener un punto de fin de conversación.

HUCB004: Como usuario, quiero que el BOT me responda una respuesta coherente cuando entienda mi intención para saber que me ha comprendido.

HUCB005: Como usuario, quiero que el BOT me indique que no me ha entendido cuando no detecte la intención de lo que le quiero decir para intentar decírselo de otra forma o saber que no es capaz de procesar esa información.

4.1.2. Épica 2: Registro del estado de ánimo

HUEA001: Como persona mayor, quiero que el BOT determine mi estado de ánimo para sentirme acompañado.

HUEA002: Como cuidador, quiero que el BOT me informe si la persona a la que cuido se siente triste o sola.

4.1.3. Épica 3: Registro de medicinas

HUMM001: Como persona mayor, quiero indicar al BOT cuándo tengo que tomar un medicamento para que pueda recordarme su toma.

HUMM002: Como persona mayor, quiero que el BOT me recuerde que me tengo que tomar un medicamento para que no se me olvide.

4.1.4. Épica 4: Citas médicas

HUMC001: Como persona mayor, quiero indicar al BOT cuándo tengo una cita médica para que pueda recordármelo.

HUMC002: Como persona mayor, quiero que el BOT me recuerde que me tengo que ir a una cita médica para poder acudir a la cita.

4.1.5. Épica 5: Detección de inactividad

HUDI001: Como cuidador, quiero que el BOT detecte inactividad de la persona mayor y le pregunte si está bien para quedarme tranquilo.

HUDI002: Como persona mayor, quiero que el BOT me inicie conversación con un saludo si llevo un tiempo inactivo, para que compruebe si estoy bien.

HUDI003: Como cuidador, quiero que el BOT me avise si, tras detectar inactividad, la persona mayor no responde, para poder intentar comunicarme yo con ella.

4.2. Historias técnicas

Los requisitos no funcionales sustentan la base tecnológica del producto. Se redactan como historias técnicas. El usuario de estas historias técnicas es el propio desarrollador.

Las historias técnicas se agrupan en diferentes épicas:

- Épica 1: Interfaz de usuario
- Épica 2: Desarrollo software
- Épica 3: Inteligencia Artificial

4.2.1. Épica 1: Interfaz de usuario

HTIU001: Como usuario, quiero poder interactuar con el BOT en lenguaje natural en idioma español, para que la interacción humano-máquina sea más fluida.

HTIU002: Como usuario, quiero que el BOT se pueda integrar en Telegram para tenerlo en el móvil sin instalar ninguna aplicación más.

HTIU003: Como usuario, quiero que el BOT me responda siempre que le escriba, para saber que me ha entendido.

4.2.2. Épica 2: Desarrollo software

HTDS001: Como desarrollador, quiero disponer de una batería de tests automáticos en el BOT que me den tranquilidad y aseguren un mínimo de calidad y robustez.

HTDS002: Como desarrollador, quiero diseñar el BOT con una arquitectura escalable para independizar la infraestructura tecnológica del dominio, de tal manera que, añadir nuevos BOTs, cambiar de base de datos o usar diferentes modelos del lenguaje, no suponga modificaciones en el dominio de la aplicación.

HTDS003: Como desarrollador, quiero disponer de una plataforma en la nube, en AWS, donde desplegar el BOT, para que esté disponible al usuario final.

4.2.3. Épica 3: Inteligencia Artificial

HTIA001: Como desarrollador, quiero disponer de una herramienta para generar un conjunto de datos en base a los diálogos, que pueda etiquetar, para entrenar un modelo capaz de predecir la intención de un usuario en base a un texto de entrada.

HTIA002: Como desarrollador, quiero disponer de una herramienta o proceso para entrenar diferentes modelos y poder comparar cuál de ellos se adecúa mejor a cada tipo de tarea.

Capítulo 5

Desarrollo de la herramienta software

El desarrollo de la herramienta software se divide en diferentes módulos. El diseño arquitectónico sigue los principios de arquitecturas limpias propuestos en (Martin, 2018). Gracias al uso de estos principios se consigue un código escalabe y preparado para pruebas automáticas, lo que facilita su mantenimiento y calidad.

El capítulo de desarrollo software se divide en las siguientes secciones:

- Estructura del proyecto
- Infraestructura local
- Infraestructura en la nube
- Herramienta de tratamiento de datos y entrenamiento
- Sistema de autenticación
- Sistema de eventos
- BOT conversacional
- Tests automáticos

5.1. Estructura del proyecto

El código fuente de todo el producto se encuentra alojado en un repositorio de **Github** (https://github.com/diegorys/bot-tfm/releases/tag/1.0.0-tfm).

El lenguaje de programación empleado ha sido **Python**. La estructura de archivos y directorios es la siguiente:

- **app:** dentro de este directorio se encuentra el subdirectorio **src**, con el código fuente de los componentes software, y el directorio **tests**, con el código para las pruebas automáticas.
- **credentials:** directorio donde se almacenan archivos de credenciales necesarias para la autenticación en servicios terceros.
- **data:** directorio donde se almacenan los diferentes conjuntos de datos generados para el entrenamiento de los diferentes modelos.
- **model:** directorio donde se almacenan los archivos **Jupyter Notebook**y archivos **Python** para la creación y el tratamiento de los conjuntos de datos y evaluación de modelos.
- **poc:** directorio para pruebas de concepto. Se emplea para ejecutar pruebas básicas de conexión con infraestructura de código, algoritmos complejos o cualquier otra necesidad que surja a la hora de desarrollo, con la finalidad de no ensuciar el código de **app** con código inservible.
- **.env_template:** fichero plantilla que contiene las claves de las variables de entorno necesarias, sin valor.
- **.gitignore:** evita que se suban al repositorio archivos auto-generados, como archivos de caché de Python, u otros archivos de credenciales.
- **Dockerfile:** contiene las instrucciones para crear las imágenes necesarias para el entorno de desarrollo.
- **README.md:** contiene una breve descripción del proyecto.
- **docker-compose.yml:** crea y orquesta los diferentes contenedores generados en base a la información de **Dockerfile**.

5.2. Infraestructura local

La utilidad de esta infraestructura es conseguir un entorno de desarrollo local que permita el desarrollo del proyecto de una forma ágil sin necesidad de desplegar en la nube.

El entorno de desarrollo levanta dos contenedores **Docker**, correspondientes a los dos servicios que se desarrollan:

5.2.1. Herramienta de tratamiento de datos y entrenamiento

Enlaza el directorio **model** como directorio de trabajo. Despliega un contenedor con una imagen de **Jupyter Notebook**. La descripción completa de esta herramienta se describe en (Sección 5.4).

5.2.2. Aplicación BOT conversacional

Enlaza el directorio **app** como directorio de trabajo. Este contenedor es una réplica de la infraestructura en la nube.

Una vez desplegado el contenedor, al acceder al mismo, se puede levantar un servicio de BOT de Telegram en local, ejecutando:

```
python .
```

El servicio se inicia y queda a la espera de que el usuario interactúe con el BOT de Telegram, tal y como se observa en la figura 5.1.

Figura 5.1: Ejecución del BOT en el entorno local.

Para pasar las pruebas automáticas (Sección 5.8) dentro del contenedor se ejecuta:

```
pytest --cov=src
```

5.3. Infraestructura en la nube

5.3.1. Descripción

Tal y como se ha indicado en (Sección 3.6), el servicio en la nube escogido para el despliegue del software es AWS. Se hace uso de Serverless para desplegar los diferentes componentes del software. En la figura 5.2 se muestra la arquitectura del BOT.

El usuario interacciona con el BOT creado en Telegram (Subsección 5.3.3). Cuando escribe al BOT, Telegram hace una petición HTTP a una URL configurada en el BOT,

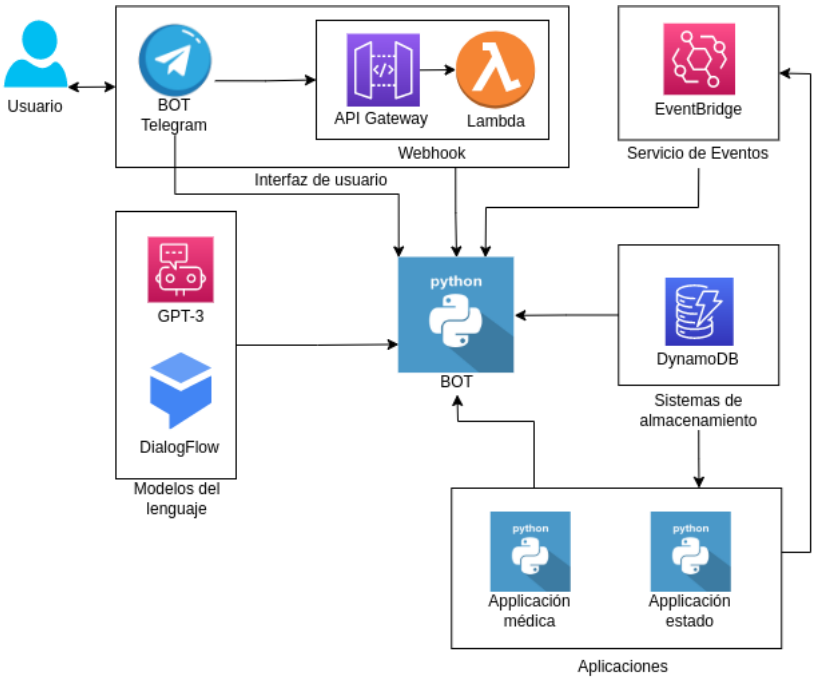


Figura 5.2: Infraestructura en la nube.

con la información del usuario emisor y el texto (o expresión de usuario). Esta petición llega a un API Gateway, que se encarga de procesarla y disparar una función Lambda. Una función Lambda es una pieza de código. En el contexto de este trabajo, es el código del BOT en lenguaje Python. El BOT almacena la expresión de usuario en una base de datos en DynamoDB para, posteriormente, poder trabajar el conjunto de datos. A continuación, pide a un modelo del lenguaje que la interprete para obtener la intención y entidades de la expresión de usuario. Si la intención se corresponde con alguna aplicación, ejecuta el comando adecuado, enviando las entidades detectadas como parámetros al comando.

Nótese que BOT no tiene ninguna dependencia externa, esto es, no conoce quién le llama ni el sistema de almacenamiento en DynamoDB, ni las aplicaciones ni modelos de lenguaje existentes. Esto se debe al uso del principio de inversión de dependencias de los principios SOLID (Martin, 2018), según el cual, el BOT define interfaces de uso que definen cómo deben comportarse los elementos de la infraestructura, como son DynamoDB o Dialogflow o las aplicaciones, quienes implementan dicha interfaz. De este modo, la adición de nuevas aplicaciones o modelos del lenguaje o el cambio de motor de base de datos, no repercute en el núcleo del código del BOT.

5.3.2. Configuración de AWS

Para poder desplegar el servicio en la nube, es necesario disponer de una cuenta en AWS. El primer paso consiste acceder a la web de AWS en https://aws.amazon.com/es/. Si se tiene cuenta en Amazon, se puede iniciar sesión. En caso contrario, será necesario registrarse.

Dentro del directorio **app** se encuentra el archivo **serverless.yml**. Ese archivo en formato yml contiene la configuración necesaria para el despliegue del servicio en AWS. Los servicios configurados en **serverless.yml** son:

- **Función Telegram:** despliega una función lambda y un API Gateway que actúa como disparador. Recibe por POST un texto del usuario, pide al BOT que lo procese y devuelve el texto de respuesta.
- **Cron:** despliega una función lambda y un evento de EventBridge que actúa como disparador. Se ejecuta cada minuto para comprobar si es necesario informar al usuario de algún evento.
- **Notify:** despliega una función lambda sin ningún disparador. Se ejecuta desde un script local para enviar mensajes directos a los usuarios a través del BOT.

Se añade el nombre del entorno a todos los recursos referidos en **serverless.yml**. De esta forma, se pueden hacer despliegues de prueba sin el temor de dejar sin servicio a los usuarios.

Para desplegar el código en AWS, se ejecuta el script **deploy.sh**, dentro del directorio **app**. En primer lugar, se debe desplegar en el entorno de *test* (ver 5.3), para verificar el funcionamiento del código desplegado sin alterar el entorno de producción. Se conserva la URL que devuelve para su posterior uso al configurar el BOT de Telegram.

```
[redacted]/app$ ./deploy.sh test

Deploying tfm to stage test (eu-west-1)

✔ Service deployed to stack tfm-test (65s)

endpoint: POST - https://[redacted].execute-api.eu-west-1.amazonaws.com/telegram
functions:
  telegram: tfm-test-telegram (23 MB)
  notify: tfm-test-notify (29 kB)

Improve API performance — monitor it with the Serverless Dashboard: run "serverless"
```

Figura 5.3: Despliegue en test.

Una vez probado que en el entorno de *test* todo funciona como debería, se procede a desplegar en producción (ver 5.4), que es el que utilizan los usuarios finales.

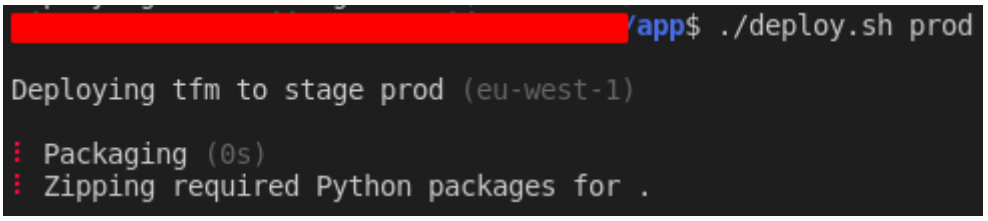


Figura 5.4: Despliegue en producción.

5.3.3. Configuración del BOT Telegram

Para crear el BOT en Telegram, se hace uso del BOT **BotFather**. En primer lugar, se busca al BOT para poder iniciar conversación con él (ver 5.5).

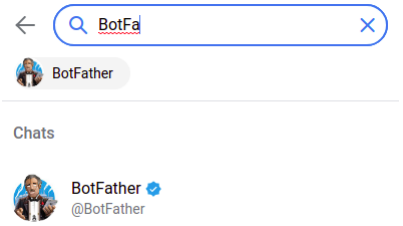


Figura 5.5: Búsqueda del BOT BotFather

A continuación, se crea un nuevo BOT, ejecutando el comando **/newbot**. BotFather guía al usuario en la creación del BOT, pidiendo datos como su nombre, tal y como se puede ver en 5.6.

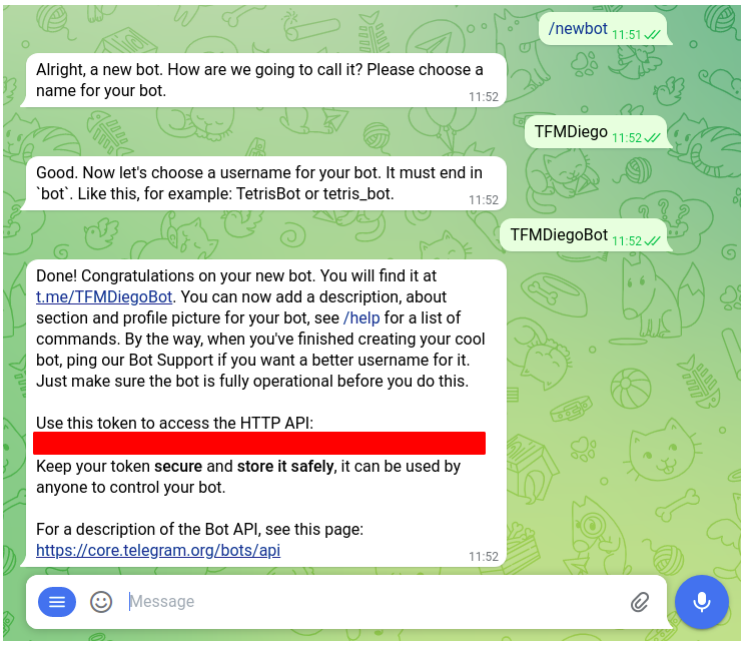


Figura 5.6: Creación de un BOT.

Finalmente, se almacena el *token* generado, como indica el tutorial, en un lugar seguro. Se utilizará este *token* para interactuar con el BOT desde el código fuente como para indicar al propio Telegram la URL a la que invocar cuando reciba un mensaje de un usuario, esto es, el **WebHook**.

El **WebHook** de Telegram es un API HTTPS al que invoca Telegram cuando el usuario envía un mensaje a un BOT. La URL de este API es la obtenida en el paso anterior (Subsección 5.3.2) en el despliegue. Del mismo modo que se despliega en AWS en un entorno de test y otro de producción, se crea un BOT diferente para cada entorno, configurando en cada uno el WebHook del entorno adecuado.

Para enlazar el BOT con su WebHook de una forma sencilla, se accede desde el navegador a la siguiente URL:

https://api.telegram.org/bot[TOKEN]/setWebhook?url=[WEBHOOK]

Siendo [TOKEN] el token obtenido en el paso anterior y WebHook la URL de WebHook mencionada anteriormente.

En este punto, ya se cuenta con dos entornos en AWS, uno para test y otro para producción, y dos BOTs que apuntan a esos dos entornos, un BOT para test y otro para producción.

5.3.4. Configuración de Dialogflow

Para trabajar con Dialogflow, es necesario tener una cuenta en Google. Una vez se dispone de dicha cuenta, se puede acceder a la consola desde la siguiente URL:

https://dialogflow.cloud.google.com/

El siguiente paso consiste en la creación de un agente conversacional, que será a quien invoque el BOT cuando el usuario le escriba un texto. En la figura 5.7 se puede apreciar cómo acceder a la creación de este agente.

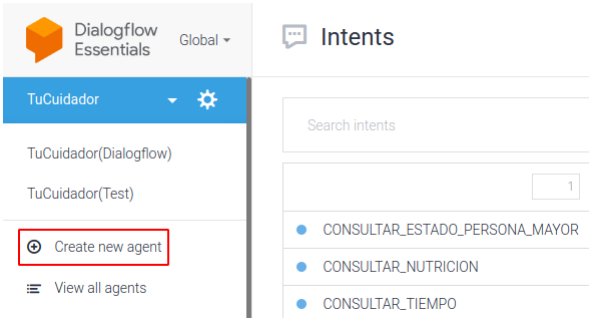


Figura 5.7: Creación de un agente conversacional en Dialogflow.

En el formulario 5.8 se rellena el nombre del agente, el idioma *Spanish - es* y la zona *Europe/Madrid*. Como proyecto de Google, se puede dejar en blanco para que se cree uno nuevo (lo más sencillo) o elegir uno ya existente. Para finalizar, se hace clic en **CREATE**.

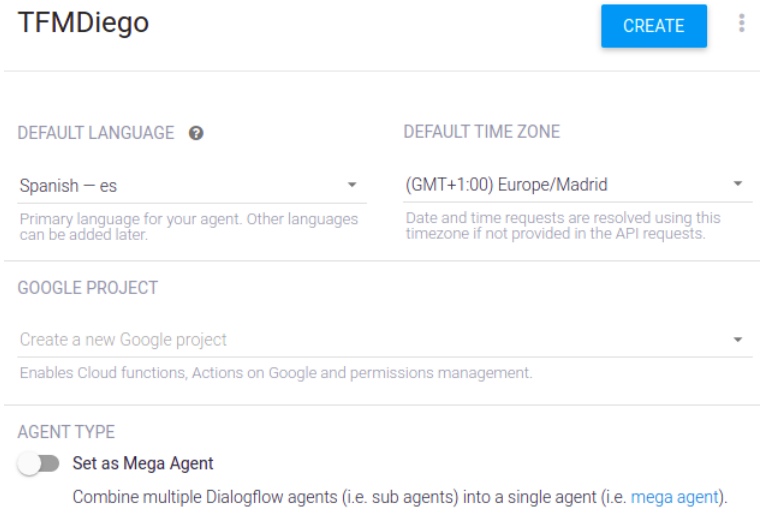


Figura 5.8: Formulario de creación de un agente conversacional en Dialogflow.

El siguiente paso es obtener un archivo de credenciales para incluir en el código fuente y así poder interactuar desde Python con el agente creado. Desde la configuración de este

agente, hacemos clic en el identificador de proyecto de Google (ver 5.9) para acceder a su configuración.

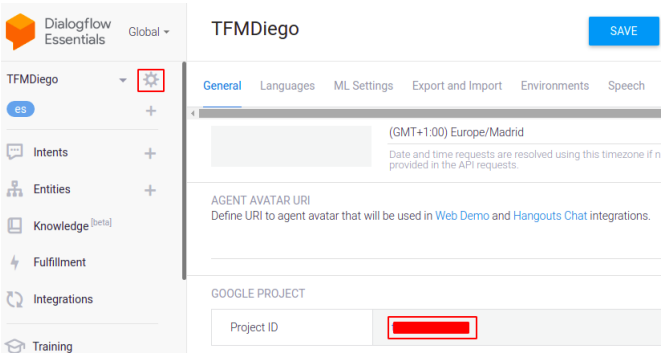


Figura 5.9: Enlace al proyecto de Google.

Desde la consola de Google, se accede a "Cuentas de servicio" (ver 5.10) para crear una nueva.

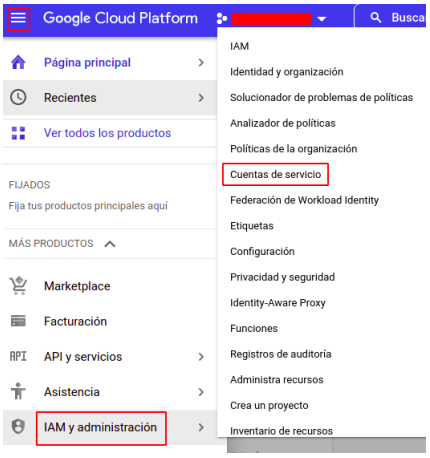


Figura 5.10: Cuentas de servicio.

Una vez creada, se accede a la cuenta. Desde la pestaña de "CLAVES" (ver 5.11) se crea una clave nueva en formato JSON.

Figura 5.11: Creación de claves para el servicio de Google.

Al finalizar, se descarga un archivo JSON con la clave creada. Este archivo se debe referenciar desde el código Python para establecer la comunicación con Dialogflow.

5.4. Herramienta de tratamiento de datos y entrenamiento

En esta sección, se definen los datos a tratar y cómo se va a trabajar con ellos para crear un conjunto de datos final y un modelo capaz de averiguar la intención del usuario en base al texto.

En primer lugar, se define la ontología con la que se van a representar las intenciones y las entidades de cada intención.

Después, se explica el proceso de generación del conjunto de datos a partir de las conversaciones mantenidas entre usuarios y BOT y cómo se transforman en los formatos necesarios para trabajar con los diferentes modelos del lenguaje.

Por último, se explica cómo se generan los diferentes modelos a aplicar al conjunto de datos para clasificarlos en la intención del usuario.

El análisis de los conjuntos de datos y resultados de los entrenamientos se detalla en la evaluación (Capítulo 6).

5.4.1. Representación del conocimiento

A partir de las expresiones de usuario obtenidas en las conversaciones entre usuario y BOT, se etiqueta manualmente la intención del usuario. Se establece un listado de intenciones a detectar, y se acota la comprensión detallada a un subconjunto de intenciones, esto es, cuando el BOT detecte una intención que reconoce y no sabe tratar, responderá diciendo que ha entendido lo que se le dice, pero que por ahora no sabe actuar, mientras

que, si sabe actuar, realizará la acción solicitada.

El conjunto total de intenciones que los usuarios han manifestado es:

- **NO_IDENTIFICADO:** se aplica a expresiones de usuario difíciles de clasificar en una intención concreta.
- **REGISTRAR_ESTADO_EMOCIONAL:** el usuario expresa al BOT su estado emocional, que queda registrado en el sistema.
- **REGISTRAR_TOMA_MEDICAMENTO:** el usuario indica que se tiene que tomar una medicación en un momento concreto.
- **REGISTRAR_DESPEDIDA:** el usuario se despide del BOT.
- **REGISTRAR_CITA_MEDICA:** el usuario registra una cita médica.
- **REGISTRAR_SALUDO:** el usuario inicia conversación saludando al BOT.
- **REGISTRAR_SITUACION_ADVERSA:** el usuario registra un problema o situación adversa.
- **REGISTRAR_ALABANZA:** el usuario se dirige al BOT para alabar sus labores.
- **REGISTRAR_DOMOTICA:** el usuario quiere interactuar con la domótica de su casa, sin hacer distinción entre si pregunta por su estado o quiere controlarlo.
- **REGISTRAR_NECESIDAD:** el usuario indica que necesita algo, pero no se concreta qué necesita.
- **REGISTRAR_CONFIRMACION:** el usuario confirma, con palabras como "de acuerdo.º "valeüna frase del BOT.
- **REGISTRAR_SINTOMA:** el usuario indica un síntoma médico, como un dolor de cabeza.
- **REGISTRAR_ANECDOTA:** el usuario cuenta una anécdota de su vida.
- **REGISTRAR_COVID:** el usuario habla del COVID, para buscar informarse o registrar que lo ha pasado.
- **CONSULTAR_ESTADO_PERSONA_MAYOR:** el cuidador pregunta cómo se encuentra la persona mayor a la que cuida.

- **REGISTRAR_MEDIDA_MEDICA:** el usuario registra un valor procedente de análisis o de algún otro dispositivo de medida, como el nivel de colesterol, la tensión o las pulsaciones.
- **REGISTRAR_ACTIVIDAD:** el usuario indica qué está haciendo, como ver la televisión o dar un paseo.
- **CONSULTAR_TIEMPO:** el usuario habla del tiempo, ya sea para preguntar si va a llover o para pedir información.
- **REGISTRAR_RECHAZO:** el usuario muestra rechazo por el BOT.
- **REGISTRAR_TELEGRAM:** el usuario pregunta alguna duda sobre Telegram.
- **ANOTAR_TOMA_MEDICAMENTO:** el usuario
- **RECORDAR_MEDICACION:** el usuario indica que se ha tomado su medicina.
- **REGISTRAR_DUDA:** el usuario pregunta dudas al BOT sobre su uso.
- **CONSULTAR_NUTRICION:** el usuario realiza alguna consulta sobre nutrición.

Las intenciones a tratar se pueden, agrupadas por dominio, ver en la figura 5.12.

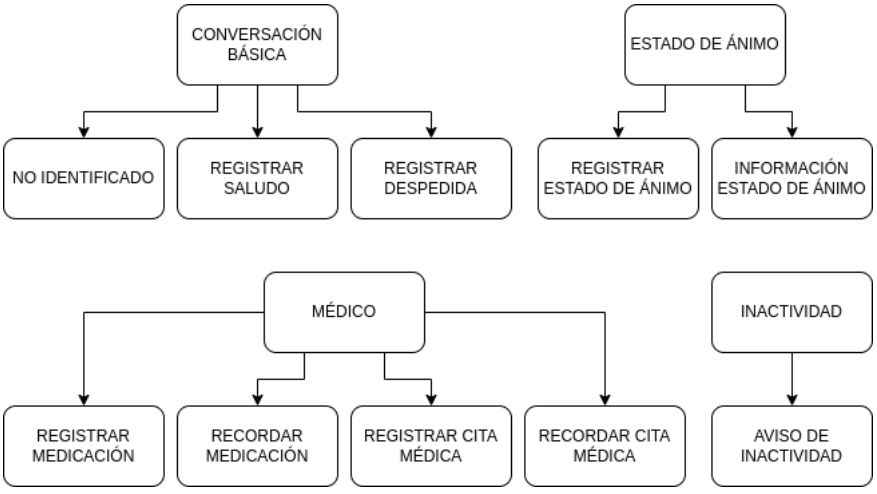


Figura 5.12: Dominios e intenciones

Para representar su ontología, se escogen *frames* con pares clave-valor. Se indican sus slots. Un slot requerido se indica con *, mientras que uno opcional se indica con ?. Se representa también el *frame* para la inactividad, que no es una intención del usuario, pero

sí se activará como intención por un sistema externo que controle la inactividad de las personas mayores.

Los dominios e intenciones que se desarrollan en este trabajo son:

- **D01 Conversación básica:** necesario para una correcta interacción entre el usuario y el BOT de una forma natural.
- **D02 Estado de ánimo:** permite identificar el estado de ánimo de un usuario (principalmente una persona mayor) e informar del estado de usuario de una persona mayor a un cuidador.
- **D03 Médico:** permite registrar tomas de medicamento y citas a usuarios (personas mayores) para que el BOT se lo recuerde.
- **D04 Inactividad:** permite avisar a cuidadores ante una detección de inactividad de sus mayores a cargo.

Conversación básica

ID0101	**Registrar saludo**
Ejemplo de entrada	¡Hola!
Entidades	usuario?
Ejemplo de respuesta	¡Hola!, ¿en qué puedo ayudarte?
Ejemplo de respuesta	¡Hola Diego!, ¿en qué puedo ayudarte?

ID0102	**Registrar despedida**
Ejemplo de entrada	¡Hasta luego!
Entidades	usuario?
Ejemplo de respuesta	¡Hasta luego!
Ejemplo de respuesta	¡Hasta luego Diego!

ID0103	No identificado
Ejemplo de entrada	¿Qué tiempo hace?
Entidades	
Ejemplo de respuesta	No entiendo lo que me dices

Estado de ánimo

ID0201	Registrar estado de ánimo
Ejemplo de entrada	Me siento bien
Entidades	estado*
Ejemplo de respuesta	Tomo nota de que estás bien

ID0202	Informar estado de ánimo negativo
Entidades	usuario-cuidador, usuario-a-cargo
Ejemplo de respuesta	Hola Diego, el estado de la persona a la que cuidas, María, es de tristeza

Médico

ID0301	Registrar toma medicamento
Ejemplo de entrada	Me tengo que tomar el ibuprofeno a las 8
Entidades	medicamento*, cuando*
Ejemplo de respuesta	Me apunto que te tienes que tomar a las 8 el ibuprofeno

ID0302	Recordar toma medicamento
Entidades	medicamento*
Ejemplo de respuesta	Te tienes que tomar el ibuprofeno

ID0303	Registrar cita médica
Ejemplo de entrada	Tengo cita con el dentista el lunes a las 7.
Entidades	cita*, cuando*
Ejemplo de respuesta	Me apunto tu cita con el dentista para el lunes a las 7.

ID0304	Recordar cita médica
Entidades	cita*, cuando*
Ejemplo de respuesta	Tienes que ir al dentista para el lunes a las 7.

Inactividad

ID0401	Aviso de inactividad
Entidades	usuario-a-cargo?
Ejemplo de respuesta	La persona a la cuidas no responde.
Ejemplo de respuesta	Tu madre no responde.

Las entidades desarrolladas en este trabajo son:

- **usuario:** usuario interlocutor. Se define con su identificador y nombre de usuario.
- **estado:** enumerado de texto, cuyos valores pueden ser triste, alegre o cansado, entre otros.
- **usuario-a-cargo:** usuario a cargo del interlocutor. Se define con su identificador y nombre de usuario.
- **medicamento:** enumerado de texto, cuyo valor es el nombre de un medicamento.
- **cuando:** unidad de tiempo que indica una fecha y hora concreta.
- **cita:** enumerado de texto, cuyo valor es el nombre de la especialidad médica, como dermatólogo o dentista.

5.4.2. Creación del conjunto de datos

Mediante la interacción de los usuarios con el BOT, se almacenan expresiones de usuario. Estas expresiones de usuario se pueden categorizar en una de las intenciones vistas.

Para crear el conjunto de datos de expresiones de usuario de personas mayores, que se utiliza para entrenar el modelo capaz de averiguar su intención, se hace uso de **Jupyter Notebook**. Desde el cuaderno se descargan los diálogos y se genera un archivo con formato JSON con la siguiente estructura:

{text: text, intent: intent, entities: {}}

En la figura 5.13 se puede ver un ejemplo de una entrada del conjunto de datos del registro de un medicamento a tomar..

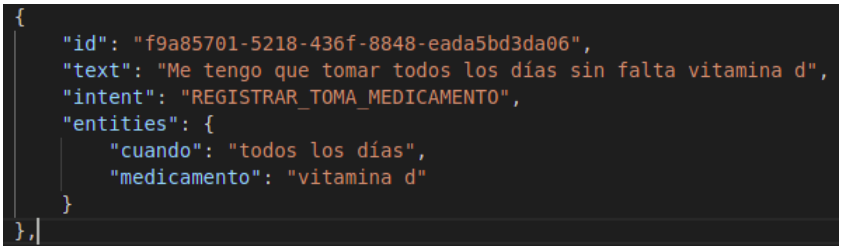


Figura 5.13: Entrada del conjunto de datos que indica la toma de un medicamento.

El código necesario se encuentra dentro de la capeta **model** del repositorio. El cuaderno se encuentra en **model/dataset_model.ipynb**.

Se adjunta como anexo (Sección A.2) el PDF extraído de **Jupyter Notebook**.

El primer capítulo del cuaderno, **Conjunto de datos**, define una serie de funciones básicas para leer las expresiones de usuario alojadas en DynamoDB y generar los conjuntos de datos en diversos formatos.

Cada vez que se crea un conjunto de datos, se le asigna una versión, con el formato "vX", siendo X un número secuencial, que empieza en 1.

Una vez se descarga la información de la base de datos, se etiquetan manualmente las intenciones y entidades, en el archivo JSON generado tras la descarga.

En cada apartado de este capítulo se visualiza una versión del conjunto de datos y se realiza un análisis de datos.

5.4.3. Entrenamiento del modelo

El segundo capítulo del cuaderno, **Entrenamiento de modelos**, se adapta el conjunto de datos al formato necesario para entrenarlo con modelo de lenguaje concreto, como Dialogflow. Se realiza un entrenamiento, o se indica cómo realizarlo de forma manual, y se determina la tasa de acierto del modelo entrenado.

En cada modelo, se determina el conjunto de datos usado y se obtiene la tasa de acierto, es decir, el porcentaje de intenciones que ha sido capaz de clasificar correctamente. Se imprimen por pantalla las que han resultado erróneamente clasificadas, a fin de poder analizar el motivo del fallo y volver a entrenar el modelo si fuera necesario.

5.4.4. Dialogflow

En (Subsección 5.3.4) se configuró el proyecto de Dialogflow. El siguiente paso consiste en subir el conjunto de datos con las intenciones, expresiones de usuario y entidades para que la IA de Dialogflow entrene el modelo. Para ello, desde la configuración del proyecto, se puede importar un proyecto que contenga toda la información necesaria. Pero primero, debe generarse este proyecto.

El código fuente que transforma el conjunto de datos en el proyecto a importar en Google se encuentra enn la ruta del repositorio **model/dialogflow**. Este proyecto es un archivo .zip que contiene una serie de archivos JSON. Para cada uno de estos archivos JSON existe un código en el directorio templates. Una vez ejecutado el código desde el cuaderno de **Jupyter Notebook** en **models/dialogflow.ipynb**, se genera, dentro de **data/dialogflow** un nuevo directorio con los archivos JSON de salida.

A continuación, se comprimen los archivos, tal y como se aprecia en la figura 5.14.

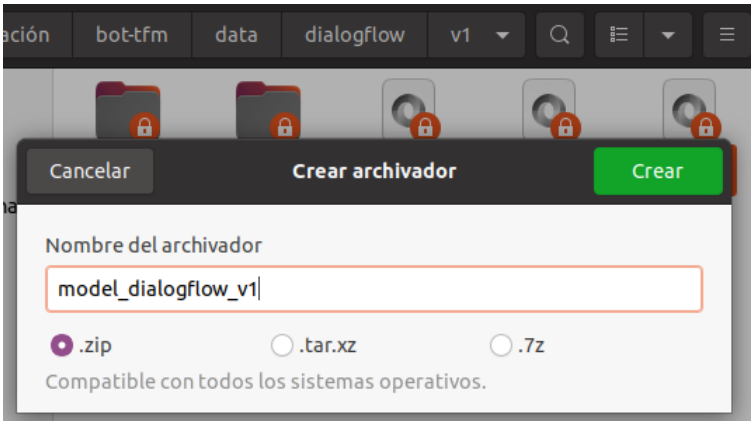


Figura 5.14: Archivos necesarios para importar un proyecto en Dialogflow.

Desde el proyecto en Dialogflow se eliminan los *intents* existentes, a excepción de *DefaultFallbackIntent*, que es la intención que se invoca por defecto cuando el modelo no ha sido capaz de determinar una intención con una confianza suficiente (se puede configurar esta confianza en la configuración del proyecto).

A continuación, se importa el proyecto comprimido anteriormente en el proyecto de Dialogflow (ver 5.15).

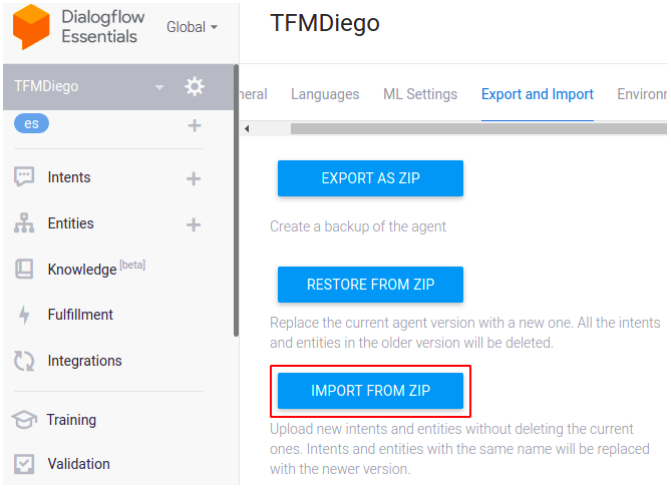


Figura 5.15: Importar un proyecto en Dialogflow.

Se elimina la intención "NO_IDENTIFICADO" para que no genere ruido, puesto que todo texto que no sea capaz de clasificar en intención dispara la intención por defecto.

Finalmente, desde el cuaderno Análisis del conjunto de datos y evaluación de modelos, se evalúa el modelo. Los resultados se encuentran en el capítulo de Evaluación (Capítulo 6).

5.5. Sistema de autenticación

La autenticación se basa en Telegram, a partir de su ID. Se ha implementado un mecanismo muy básico, puesto que no es objetivo del proyecto el sistema de autenticación. En la figura 5.16 se puede ver diseño arquitectónico del sistema de autenticación. En el código se corresponde con el módulo **sso**, Single Sign On (SSO).

Siguiendo los principios de arquitectura de puertos y adaptadores descritos en (Martin, 2018), en el dominio encontramos una clase que representa a un usuario y una interfaz de repositorio que define los métodos necesarios para gestionar a los usuarios, mientras que en la infraestructura encontramos una implementación en DynamoDB del repositorio de usuarios.

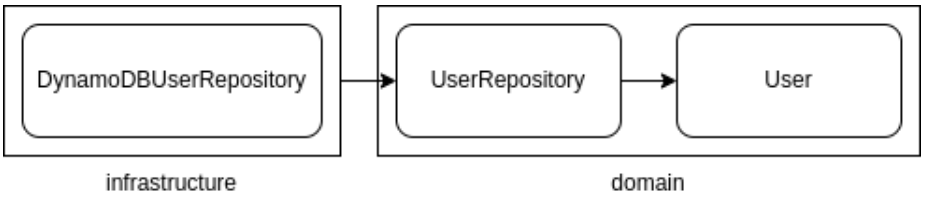


Figura 5.16: Sistema de autenticación.

Se comporta como un paquete que importan los diferentes módulos.

5.6. Sistema de eventos

Sistema encargado de registrar eventos a notificar al usuario. Las diferentes aplicaciones se conectan con el servicio de avisos para indicarle qué avisos tiene que dar al usuario. Como se muestra en la figura 5.17, sigue la misma arquitectura vista en el módulo SSO.

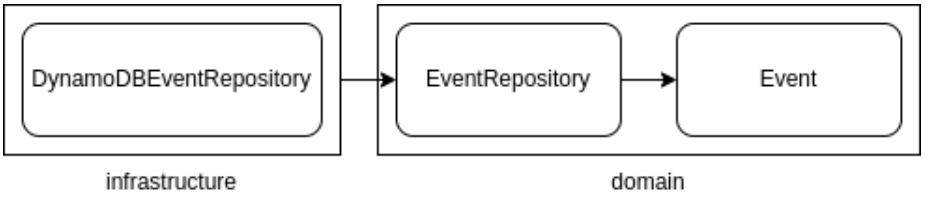


Figura 5.17: Sistema de eventos.

Se comporta como un paquete que importan las aplicaciones.

5.7. BOT conversacional

El BOT, pese a ser el centro del proyecto, como componente software no es más que un detalle técnico (Martin, 2018) en la arquitectura hexagonal, o de puertos y adaptadores, empleada en el diseño arquitectónico del BOT conversacional. Actúa como interfaz de usuario para el resto de aplicaciones, del mismo modo que lo haría una página web o una aplicación móvil.

En la figura 5.18 se muestra la arquitectura del BOT.

Se describen a continuación los diferentes componentes de la arquitectura del BOT conversacional.

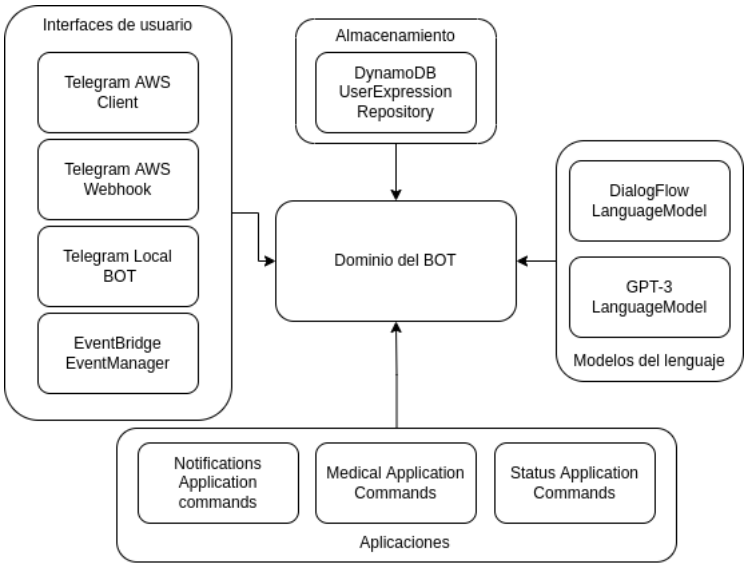


Figura 5.18: Arquitectura del BOT: dependencias entre infraestructura y dominio

5.7.1. Dominio del BOT

El componente de dominio del BOT es un paquete de clases encargado de orquestar una comunicación entre una interfaz (usuario o sistema de eventos) y el BOT, desde que la interfaz envía una orden al BOT hasta que éste la procesa y emite una respuesta al usuario.

Este paquete es lo suficientemente genérico como para no conocer el propósito del BOT, es decir, las aplicaciones que a él se conectan, las interfaces desde las que se le puede llamar, los diferentes modelos del lenguaje que puede utilizar o dónde y cómo se almacena la información.

Al igual que el SSO y el sistema de eventos ya descritos, se diseña una arquitectura de puertos y adaptadores, centrada en el dominio, representado por la clase **ReactiveBOT**.

En la figura 5.19 se presenta la relación entre las clases del dominio.

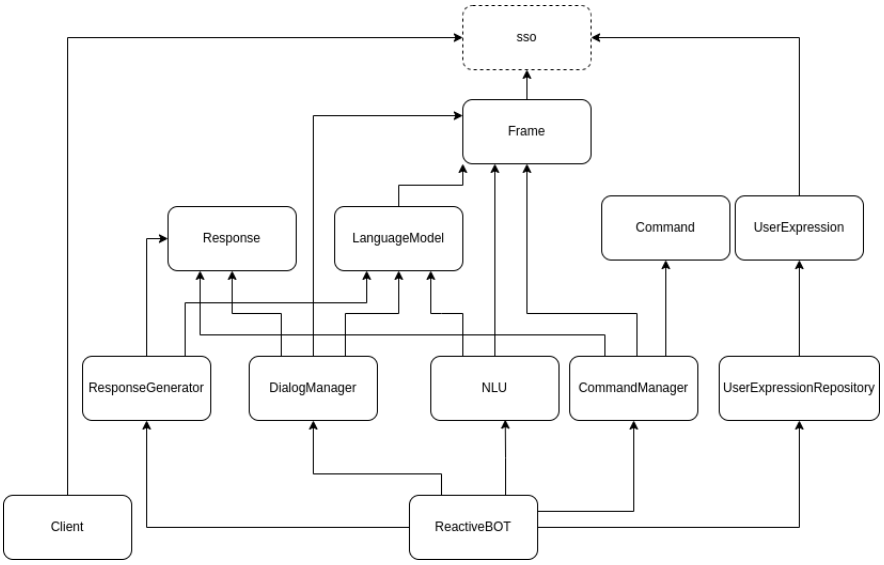


Figura 5.19: Diagrama de componentes del Dominio del BOT.

Se establecen una serie de interfaces que se implementan en la infraestructura, a modo de adaptadores primarios (interfaces) o secundarios (conexión con sistemas terceros, como modelos del lenguaje o repositorios de almacenamiento).

El dominio permanece inmutable, puesto que no se esperan modificaciones sobre él. Los cambios que se pueden producir quedan reflejados en la infraestructura.

En el dominio existe una alta cohesión entre sus diferentes clases y un bajo acoplamiento con otros módulos, referenciando tan sólo al módulo sso para obtener información del usuario.

Frame

La entidad **Frame** representa una intención y las entidades de la misma. Tiene una referencia al usuario asociado a la entidad.

Modelo del lenguaje

La interfaz **LanguageModel** especifica qué métodos deben tener los modelos del lenguaje Subsección 5.7.3 usados en el BOT para resolver las diferentes tareas de PLN requeridas por los diferentes componentes del BOT. En la figura 5.20 se describe la interfaz.

```python
from src.conversational_bot.frame import Frame


class LanguageModel:
    def identifyIntent(self, text: str):
        pass

    def generateRequireParametersText(self, frame: Frame) -> str:
        pass

    def generateText(self, frame: Frame) -> str:
        pass
```

Figura 5.20: Interfaz LanguageModel.

Los métodos que debe implementar un modelo de lenguaje son:

- **identifyIntent:** recibe una cadena de texto con la expresión del usuario. El modelo predice la intención (tarea de clasificación) y determina las entidades (tarea de extracción de entidades). Devuelve la intención del usuario como cadena de texto y un diccionario de entidades y valores.
- **generateRequireParametersText:** recibe un *frame* incompleto, es decir, sin todos los campos requeridos. Devuelve una cadena de texto generada por el modelo (tarea de generación de texto) que expresa la intención de pedir al usuario los datos faltantes.
- **generateText:** recibe un *frame* completo. El modelo realiza la tarea de generación de texto para devolver una cadena de texto asociada al frame recibido.

Client

La interfaz *Client* (Cliente) ofrece un método para emitir un mensaje como cadena de texto a un usuario.

Actúa como un puerto primario del BOT, al que se conectan interfaces que no son de usuarios, como un cron de eventos, para que el BOT se pueda comunicar de forma proactiva.

En la infraestructura del proyecto se ubican las clases que implementan esta interfaz.

Response

La entidad **Response** representa una respuesta destinada al usuario final. Tiene una intención y un texto generado por el modelo de lenguaje.

Command

La interfaz **Command** se comporta como un puerto secundario para establecer una comunicación con aplicaciones terceras.

Están asociados a una intención y tienen unos parámetros, que son las entidades de la intención.

En **Aplicaciones** (Subsección 5.7.5) se explica el funcionamiento de los comandos con más detalle.

UserExpression

Una expresión de usuario representa un texto emitido por un usuario y recibido por el BOT.

Sus atributos son el texto, el usuario emisor, la intención detectada por el BOT, las entidades asociadas a la intención, la respuesta a emitir al usuario y la fecha en la que se generó la expresión.

UserExpressionRepository

Repositorio de expresiones de usuario. Interfaz que ofrece un método para guardar una expresión de usuario.

En la infraestructura del BOT se debe definir una clase que implemente esta interfaz.

NLU

El BOT responde a varias intenciones. Por tanto, la primera tarea del componente **NLU** (*Natural Language Understanding*, Comprensión del Lenguaje Natural) consiste en identificar la intención del usuario a partir del texto de entrada. Para ello, el componente **NLU** hace uso del modelo del lenguaje recibido en el constructor, quien realiza esta tarea de procesamiento de lenguaje natural de clasificación de texto.

El modelo de lenguaje devuelve el *frame*, con la intención determinada y sus entidades asociadas.

Finalmente, el componente **NLU** devuelve el frame, que representa la semántica de la frase expresada por el usuario.

DialogManager

El frame proporcionado por el componente **NLU** puede estar incompleto porque el usuario no haya proporcionado toda la información necesaria.

Por este motivo, el componente **DialogManager** (gestor de diálogo) se encarga de generar un contexto del *frame* actual y pedir al usuario la información restante, hasta completar el *frame*.

La salida de este componente es un *frame* completo, que representa la semántica de la frase expresada por el usuario.

CommandManager

Una vez conseguido un *frame* completo, el componente **CommandManager** (gestor de comandos) se encarga de buscar el comando asociado a una intención y ejecutarlo, pasando como parámetros las entidades extraídas de la expresión de usuario.

Recibe como entrada el *frame*, que contiene la intención del usuario, y determina si existe una acción asociada a ella. Si es así, la ejecuta y genera un *frame* con el resultado de la ejecución. En caso de error, genera un *frame* indicando cuál es el error.

ResponseGenerator

Por ultimo, el componente **ResponseGenerator** (generador de respuestas) recibe el *frame* procedente del gestor de comandos para generar una respuesta en lenguaje natural. Para ello, utiliza el modelo de lenguaje recibido como dependencia para que ejecute la tarea de generación de texto en base al frame.

ReactiveBOT

El componente **ReactiveBOT** (BOT reactivo) se encarga de procesar una expresión de usuario y devolver una respuesta. Es necesario que esta interacción sea automática y no por un sistema de eventos para asegurar que el usuario siempre obtiene respuesta.

En su constructor, a este BOT se le inyectan los componentes **NLU**, **DialogManager**, **ResponseGenerator**, **CommandManager** y **UserExpressionRepository**.

En la figura 5.21 se describe el flujo completo del BOT reactivo.

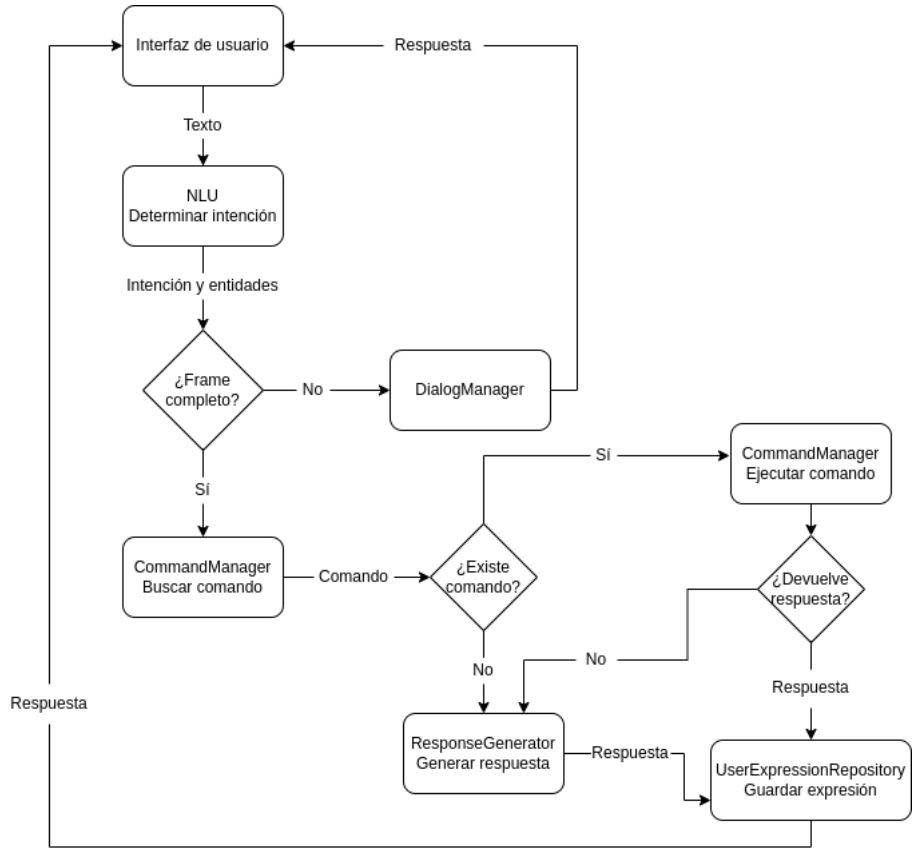


Figura 5.21: Diagrama de flujo del BOT Reactivo.

Cuando una interfaz de usuario invoca al BOT, le envía un objeto **BOT reactivo** (del módulo SSO), el texto correspondiente a la expresión emitida por el usuario y la fecha de emisión.

Lo primero que hace el BOT es determinar la intención y las entidades de la expresión de usuario, para lo que invoca al componente **NLU**.

Si el frame devuelvo por el componente NLU no está completo, se invoca al componente **DialogManager** para que se encargue de preguntar al usuario por los datos requeridos por el *frame*.

Si el *frame* devuelto por el componente **NLU** es completo, se invoca al componente **CommandManager** para que ejecute el comando adecuado, en caso de existir.

A continuación, se devuelve una respuesta, ya sea proporcionada por el comando asociado a la intención, o si no existe, generada por el componente **ResponseGenerator**.

Se almacena en el repositorio de expresiones de usuarios la expresión del usuario generada y se devuelve la respuesta a la interfaz de usuario que invocó al BOT.

5.7.2. Interfaces

Según la arquitectura descrita, las interfaces son componentes externos que interactúan con el BOT. Se conectan a los dos puertos primarios que ofrece, que son el BOT reactivo, para interfaces de usuario, y el BOT proactivo, para el resto de interfaces.

La adición de nuevas interfaces que hagan uso del BOT utilizando sus puertos primarios implican sólo la creación de estos módulos de interfaz.

Telegram

Telegram tiene tres usos distintos en la aplicación:

- **TelegramClient:** implementa la interfaz Client descrita en el dominio del BOT, para que el BOT proactivo pueda comunicarse con el usuario sin que éste inicie una conversación.
- **BOT Local:** instancia un BOT de Telegram en el entorno local para facilitar el desarrollo del proyecto. Invoca al BOT reactivo.
- **WebHook:** corresponde a la URL configurada del API Gateway. Invoca al BOT reactivo.

EventBridge EventManager

Interfaz que se conecta al cliente que actúa como BOT proactivo, al que envía un mensaje cuando tenga que avisar al usuario de un evento.

5.7.3. Modelos del lenguaje

Los modelos del lenguaje se comportan como adaptadores secundarios. la adición de nuevos modelos del lenguaje implica únicamente la creación de un módulo que implemente la interfaz LanguageModel del dominio.

Este proyecto utiliza el modelo de lenguaje **Dialogflow**. Gracias a la arquitectura del BOT, es posible añadir otros modelos del Estado del arte, como **GPT-3** o **MarIA**, simplemente implementando la ya descrita interfaz **LanguageModel**.

Esta arquitectura permite que cada componente del BOT (NLU, Gestor del Diálogo, Generador de Respuestas) emplee un modelo del lenguaje, es decir, no tienen por qué emplear todos el mismo.

El modelo **GPT-3** se comenzó a utilizar en las primeras fases del desarrollo, pero, dado que su uso tiene un coste asociado, se abandonó su uso en favor de Dialogflow. No obstante, se describe la parte implementada de **GPT-3**.

Dialogflow

En el repositorio de código, dentro de **app/src/language_models/dialogflow**, se encuentra la clase **DialogFlowLanguageModel** (ver 5.22). Esta clase implementa los métodos definidos en la interfaz **LanguageModel**.

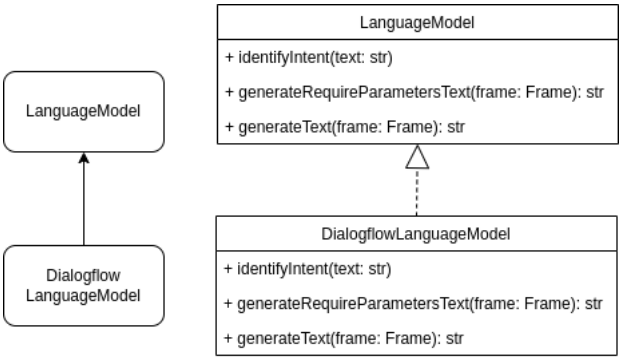


Figura 5.22: Modelo del lenguaje Dialogflow

Para que esta clase sea capaz de comunicarse con el servicio en la nube de Google Dialogflow, es necesario configurar en el repositorio la siguiente información del proyecto:

- **PROJECT_ID:** identificador del proyecto de Google.
- **SESSION:** sin uso, se puede poner un número aleatorio.
- **LANGUAGE_CODE:** código de idioma del agente. Se corresponde con **es**.
- **GOOGLE_APPLICATION_CREDENTIALS:** variable de entorno que debe almacenar la ruta al archivo de credenciales en formato JSON.

La implementación del método **identifyIntent** se comunica con **Google Dialogflow** haciendo uso del paquete tercero **google-cloud-dialogflow**, establecido en el archivo de requisitos de Python del repositorio (requirements.txt).

La implementación de los métodos **generateRequireParametersText** y **d**evuelve la última respuesta del agente.

GPT-3

En la carpeta **poc/gpt3** del repositorio se ubica **GPT3LanguageModel**. Esta clase implementa los métodos definidos en la interfaz **LanguageModel**.

Para que sea capaz de comunicarse con el servicio en la nube de OpenAI, donde se encuentra GPT-3, es necesario configurar el **API Key** de **GPT-3**. Se utiliza el paquete tercero **openai** para esta comunicación, a quien se provee del **API Key**.

La implementación de **identifyIntent** se corresponde con una tarea de **Completion de OpenAI**.

Una vez determinada la intención, se invoca una acción concreta para la extracción de entidades, que hace uso de **Completion de OpenAI**. Se ha desarrollado de forma manual una acción por cada intención a reconocer por parte del BOT.

Esta acciones, a su vez, tienen un método capaz de generar texto de vuelta al usuario, invocando una tarea de **Completion de OpenAI**.

Por cuestiones económicas, no se pudo seguir con el desarrollo con GPT-3, pese a los buenos resultados que empezaba a ofrecer, como se verá en la evaluación (Capítulo 6), al ser, como se ha indicado previamente, un servicio de pago.

5.7.4. Sistemas de almacenamiento

El sistema de almacenamiento del BOT ofrece una interfaz que define un repositorio de expresiones de usuario que se ofrece en el dominio del BOT.

Esta interfaz se implementa en la infraestructura.

DynamoDB

Implementación de la interfaz **UserExpressionRepository** en **DynamoDB**, dentro de AWS. Guarda las expresiones de usuario en una tabla en **DynamoDB**.

5.7.5. Aplicaciones

Las aplicaciones, desde el punto de vista software, son el centro de la arquitectura. En este proyecto, el centro es el BOT conversacional y las aplicaciones se comportan como infraestructura que enganchan a modo de *plug-in* con el BOT conversacional.

Para añadir una nueva aplicación, se crea un módulo en el repositorio de código, dentro de **app/src/applications**. Cada módulo tiene una serie de comandos, casos de

uso, dominio e infraestructura. Los comandos implementan la interfaz Command del dominio. Las aplicaciones se comportan como puertos secundarios. La figura 5.23 refleja la arquitectura de una aplicación.

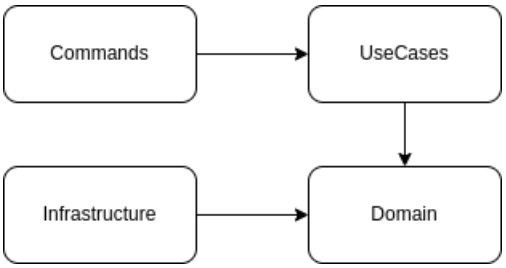


Figura 5.23: Arquitectura de una aplicación

Las aplicaciones tienen:

- **Comandos:** puertos secundarios que invocan a los casos de uso de la aplicación.
- **Casos de uso:** contiene los casos de uso de la aplicación.
- **Dominio:** contiene la lógica de negocio de la aplicación.
- **Infraestructura:** contiene la implementación de la infraestructura necesaria para la aplicación.

Aplicación médica

La aplicación médica se encarga del cuidado de la salud de sus usuarios.
La figura 5.24 refleja la arquitectura de la aplicación médica.

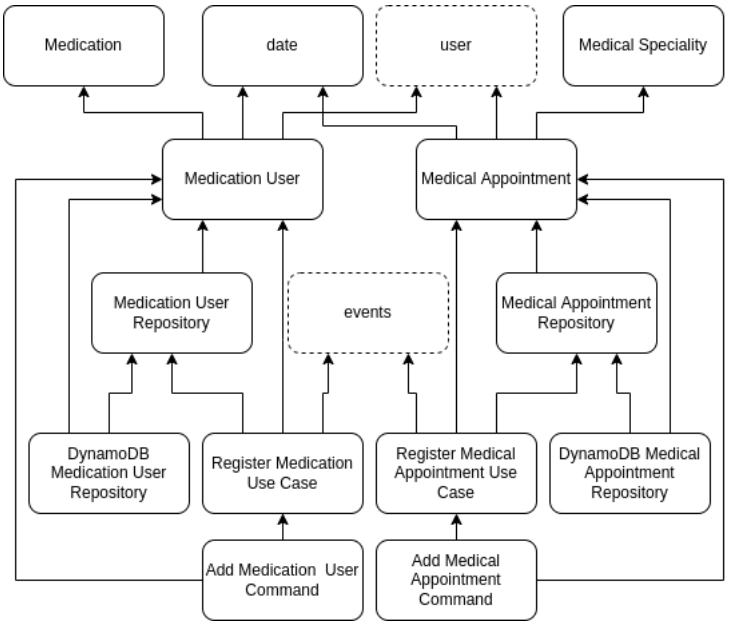


Figura 5.24: Arquitectura de la aplicación médica

Los componentes que se pueden encontrar en la arquitectura son:

- **Medication:** entidad medicamento, que contiene su nombre.
- **MedicalSpeciality**: entidad especialidad médica, que contiene el nombre de la especialidad.
- **Date:** *value object* que representa un momento en el tiempo.
- **MedicationUser:** entidad contiene un medicamento, un usuario y el momento en el que se tiene que tomar el usuario la medicación.
- **MedicalAppointment:** entidad cita médica con los datos del usuario y de la especialidad médica.
- **MedicationUserRepository:** interfaz que define los métodos que debe tener un repositorio de medicación de usuario.
- **MedicalAppointmentRepository:** interfaz que define los métodos que debe tener un repositorio de citas médicas.
- **RegisterMedicationUseCase:** registra la necesidad de un usuario de tomar un medicamento en un momento determinado.

- **RegisterMedicalAppointmentUseCase:** registra una cita del usuario con un médico.
- **AddMedicationUserCommand:** comando que implementa **Command** del BOT. Invoca el caso de uso **RegisterMedicationUseCase**. Envía un mensaje al gestor de eventos para que cree un nuevo recordatorio.
- **AddMedicalAppointmentCommand** comando que implementa **Command** del BOT. Invoca el caso de uso **RegisterMedicalAppointmentUseCase**. Envía un mensaje al gestor de eventos para que cree un nuevo recordatorio.
- **DynamoDBMedicationUserRepository:** almacena la medicación del usuario en **DynamoDB**. Implementa la interfaz **MedicationUserRepository**.
- **DynamoDBMedicalAppointmentRepository:** almacena las citas médicas del usuario en **DynamoDB**. Implementa **MedicalAppointmentRepository**.

Hace uso del paquete **sso** para obtener los datos del usuario y del paquete **events** para añadir nuevos eventos.

Aplicación estado

La aplicación de estado registra el estado de un usuario e informa del estado de una persona mayor a un cuidador asociado.

La figura 5.25 refleja la arquitectura de la aplicación de estado.

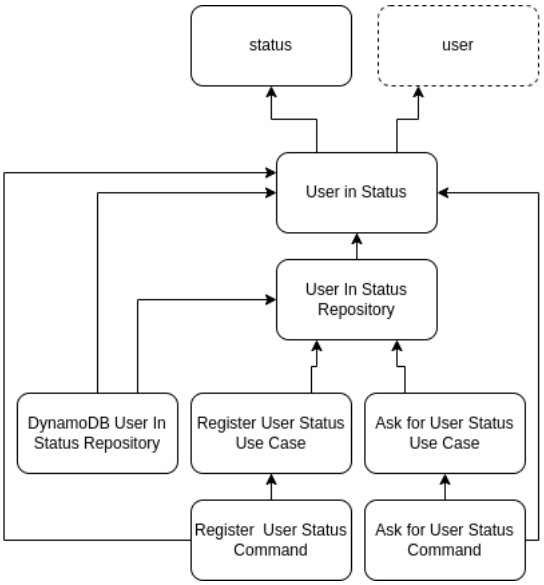


Figura 5.25: Arquitectura de la aplicación estado

Los componentes que se pueden encontrar en la arquitectura son:

- **Status:** entidad estado, que contiene su nombre.
- **UserInStatus:** entidad que indica el estado en el que se encuentra un usuario. Se almacena la fecha en la que se registra el estado.
- **UserInStatusRepository:** interfaz que define los métodos que debe implementar un repositorio de estados de usuario.
- **RegisterUserStatusUseCase:** registra el estado de un usuario.
- **AskForUserStatusUseCase:** consultado el estado de un usuario.
- **RegisterUserStatusCommand** comando que implementa **Command** del BOT. Invoca el caso de uso **RegisterUserStatusUseCase**.
- **AskForUserStatusCommand** comando que implementa **Command** del BOT. Invoca el caso de uso **AskForUserStatusUseCase**.
- **DynamoDBUserInStatustRepository:** almacena el estado actual del usuario en **DynamoDB**. Implementa **UserInStatusRepository**.

Hace uso del paquete **sso** para obtener los datos del usuario.

Aplicación inactividad

Aplicación que comprueba periódicamente la última vez que un usuario interactuó con el BOT para enviarle un mensaje si se detecta una actividad superior a tres horas.

La figura 5.26 refleja la arquitectura de la aplicación de inactividad.

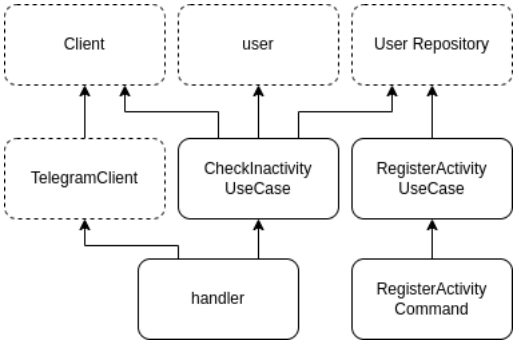


Figura 5.26: Arquitectura de la aplicación inactividad

Los componentes que se pueden encontrar en la arquitectura son:

- **RegisterActivityUseCase:** registra actividad del usuario.
- **CheckInactivityUseCase:** consultado la última actividad del usuario, para, si es superior a tres horas, preguntarle cómo está. La siguiente vez que se ejecuta el caso de uso, si el usuario sigue sin haber respondido, envía un mensaje a su cuidador.
- **RegisterActivityCommand** comando que implementa **Command** del BOT.

  Invoca el caso de uso **RegisterActivityUseCase** cada vez que el usuario interactúa con el BOT.
- **handler** manejador de eventos, subscrito a un cron que se ejecuta tres veces al día (a las 12:00, a las 16:00 y a las 20:00). En cada ejecución, invoca el caso de uso **CheckInactivityUseCase**.

5.8. Pruebas automáticas

A fin de evaluar y verificar el correcto funcionamiento del software, se realizan pruebas automáticas. Para ello, se utiliza el paquete **pytest** de Python.

Con la inclusión de pruebas automáticas, se miden dos aspectos de la calidad del software:

- **Cobertura de código:** mide el porcentaje de código fuente cubierto por pruebas automáticas.
- **Porcentaje de pruebas superado:** mide el porcentaje de pruebas que se han superado.

Para ejecutar las pruebas automáticas, se debe lanzar el comando **pytest** desde la carpeta **app** o directamente desde dentro del contenedor.

En la figura 5.27 se muestra un ejemplo de ejecución de pruebas. Se aprecia que la cobertura de código es del 24 %. El número de tests superado es 8, se ha omitido la ejecución de un test, por tanto, el porcentaje de tests superado es del 88.89 %. Este resultado no cumpliría los aspectos de calidad definidos.

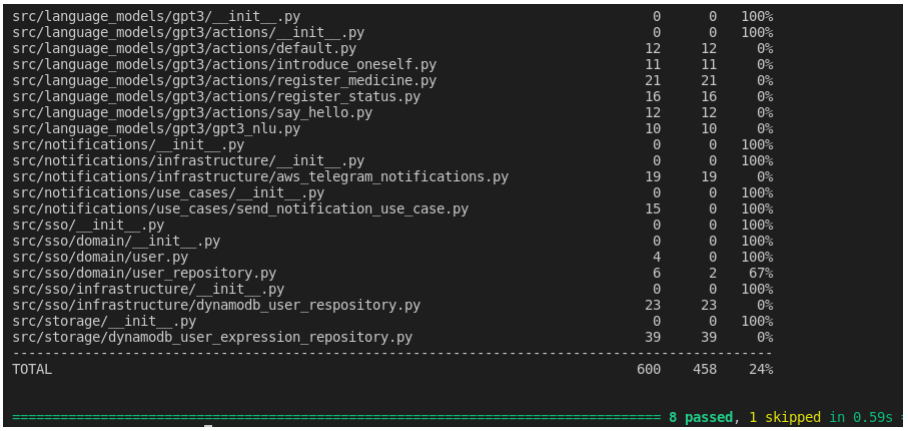


Figura 5.27: Ejemplo de resultado de ejecución de pruebas automáticas

Los resultados se muestran en el capítulo siguiente (Capítulo 6), **Evaluación**.

Capítulo 6

Evaluación

En el siguiente capítulo, se evalúan, según se indicó en (Subsección 2.1.3) diferentes aspectos del trabajo, a fin de determinar el grado de cumplimiento de los diferentes objetivos (Capítulo 3) planteados al inicio del proyecto.

6.1. Análisis del conjunto de datos y evaluación de modelos

En el cuaderno que se encuentra en el anexo (Sección A.2) se describe el proceso de análisis del conjunto de datos y evaluación de modelos.

Se consiguen detectar las necesidades de los usuarios, tras el etiquetado manual de un conjunto de datos de 236 diálogos, como se puede ver en la figura 6.1.

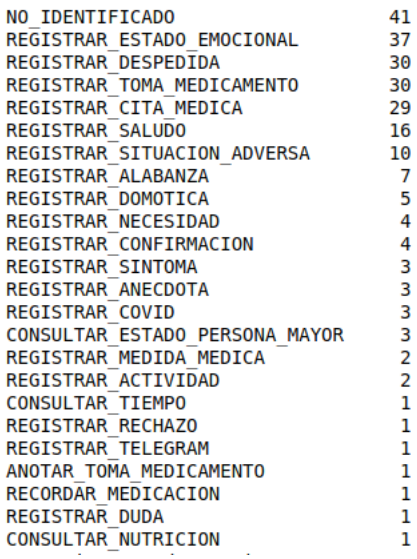


Figura 6.1: Necesidades detectadas

De los 236 diálogos, 41 entradas se etiquetan como no identificadas, al no quedar clara la intención del usuario durante el etiquetado manual. Se conservan para su uso como ejemplos negativos. Se detectan 25 necesidades diferentes.

De esas necesidades, se obtiene un conjunto de datos que cubre las intenciones recogidas en los requisitos, esto es, registrar el estado emocional, la toma de medicamentos y citas médicas, además de las intenciones básicas de saludo y despedida.

Dado que se han podido descubrir las principales necesidades de los usuarios, **se considera cumplido el objetivo** "Descubrir necesidades reales de las personas mayores a partir de la base de datos de diálogos y clasificarlas en intenciones según el diálogo".

A partir de este conjunto de datos, se genera un subconjunto de entrenamiento con el 80 % de datos para Dialogflow y otro subconjunto de un 20 % de datos para validación. Se sube el conjunto a Dialogflow y se entrena desde la plataforma. Se valida con el conjunto de validación. Se comprueba que se ha alcanzado el **89.66 % de tasa de acierto** (ver 6.2) al identificar la intención del usuario, **superando el 85 % definido en el objetivo**.

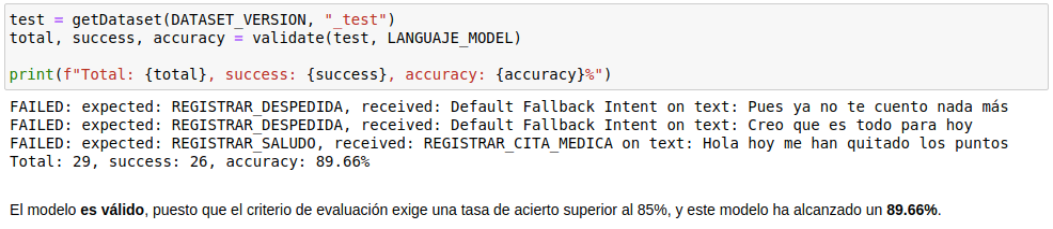


Figura 6.2: Evaluación del conjunto de datos y del modelo

Por tanto, **se considera cumplido el objetivo** "Elaborar un conjunto de datos que permita entrenar un modelo con una tasa de acierto superior al 85 % en la tarea de detección de la intención".

6.2. Investigar modelos preentrenados y plataformas PLN

El objetivo "Investigar los diferentes modelos preentrenados y plataformas de PLN que se pueden usar para entrenar al BOT conversacional" **se ha cubierto** al desarrollar el Estado del Arte, tal y como se puede ver en (Subsección 2.1.2).

6.3. Calidad del producto

El desarrollo del BOT se valida comprobando que existe un BOT en Telegram, accesible desde **@CuidadorMayoresBot**.

Tras la integración del modelo del lenguaje en el BOT, se pueden observar en la figura 6.3 diferentes reacciones. En la primera frase, el BOT entiende la intención del usuario de registrar su estado y sabe reaccionar, indicando que lo ha registrado. En la segunda frase, el BOT ha sido capaz de identificar que el usuario quiere registrar la activada que está realizando, pero aún no ha sido desarrollado el comando para su registro, y se lo indica al usuario. En la tercera frase, el BOT no es capaz de identificar la necesidad del usuario.

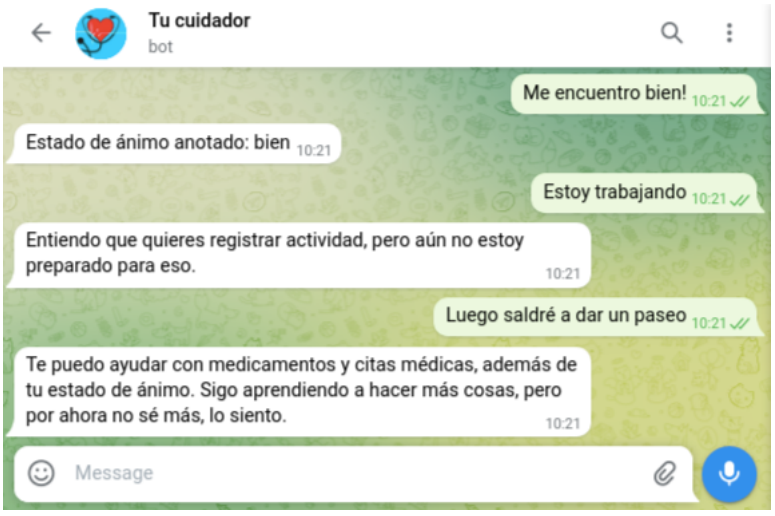


Figura 6.3: Respuestas del BOT ante diferentes situaciones

Se verifica que responde a las intenciones definidas en los requisitos de saludo, registro de medicación, cita médica, estado de ánimo y despedida interactuando con el BOT, como se puede ver en la figura 6.4.

Figura 6.4: Respuesta a las intenciones identificadas

Se verifica que **se ha cumplido el objetivo** "Desarrollar un BOT conversacional empleando las bibliotecas de terceros que sean necesarias, que utilice el modelo entrenado y actúe en consecuencia de la intención identificada".

Para garantizar que el usuario final no se ve afectado por fallos del sistema y siempre recibe una respuesta, se realiza una captura de excepciones ante situaciones no controladas, que devuelvan información al usuario (6.5). Ante un error técnico no controlado, el sistema informa al usuario (6.5a). Ante el uso del BOT por un usuario no dado de alta, se informa al usuario (6.5b).

(a) Respuesta a un error técnico no controlado (b) Usuario no reconocido

Figura 6.5: Captura de errores

Para validar la calidad del software y su rosbustez, se realizan pruebas automáticas. En la figura 6.6 se muestra el resultado su ejecución. Se aprecia cómo se ha alcanzado un 95 % de cobertura de código y se han superado 54 pruebas automáticas sin ningún fallo.

```
src/events/domain/event.py                                    22      3    86%   10, 15, 20
src/events/domain/event_repository.py                         13      0   100%
src/events/use_cases/remember_use_case.py                     29      0   100%
src/factories/__init__.py                                      0      0   100%
src/factories/command_manager_factory.py                      50      0   100%
src/language_models/__init__.py                                0      0   100%
src/language_models/dialogflow/__init__.py                     0      0   100%
src/language_models/dialogflow/dialogflow_language_model.py   57      5    91%   38-43
src/notifications/__init__.py                                  0      0   100%
src/notifications/use_cases/__init__.py                        0      0   100%
src/notifications/use_cases/send_notification_use_case.py     15      0   100%
src/sso/__init__.py                                            0      0   100%
src/sso/domain/__init__.py                                     0      0   100%
src/sso/domain/user.py                                        66      4    94%   54, 59, 64, 69
src/sso/domain/user_repository.py                             27     11    59%   15-25
-----------------------------------------------------------------------------------------------
TOTAL                                                        672     36    95%

============================== 54 passed in 6.39s ==============================
root@fa5d84afdae3:/app#
```

Figura 6.6: Resultado de ejecución de pruebas automáticas

Dado que el objetivo era cubrir un mínimo de un 90 % de código sin fallos, se considera que el **objetivo está superado**.

Gracias al tratamiento de excepciones y las pruebas automáticas, se considera que **se ha cumplido** el objetivo "Dotar al sistema de robustez para recuperarse de errores sin que el usuario final se vea afectado".

Se comprueba que el BOT es capaz de iniciar una conversación de forma proactiva.

Para ello, se interactúa con el BOT, como se puede observar (ver 6.7). Se aprecia que, cuando cumple la hora indicada, con un margen de 15 minutos de antelación, el BOT inicia una conversación para recordar al usuario que tiene que tomarse el medicamento indicado o acudir a una cita médica.

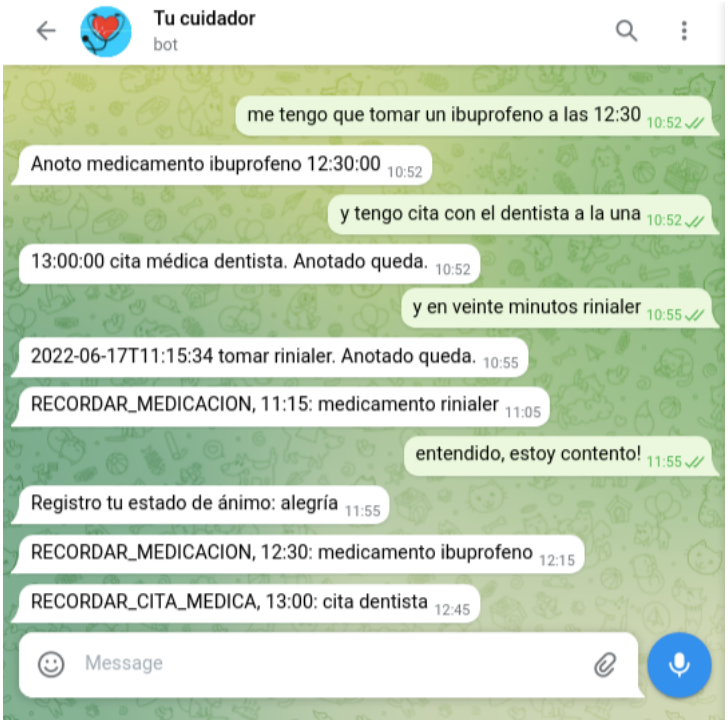


Figura 6.7: El BOT recuerda al usuario su medicación y citas médicas

La arquitectura software definida en Subsección 5.7.1 permite, gracias a la inversión de dependencias, la creación de nuevos componentes software sin que afecte al dominio de la aplicación.

En base a las pruebas anteriores, **se considera cumplido el objetivo** "Diseñar la arquitectura de un BOT conversacional que almacene información de usuarios y sea capaz de hablar de forma proactiva ante determinados eventos, de forma escalable, que permita al software crecer mediante la inclusión de nuevos componentes, según aumente el número de funcionalidades a implementar en base a las necesidades de mayores y jóvenes. La arquitectura debe estar preparada para el uso futuro del BOT conversacional desde plataformas diferentes a Telegram.".

El objetivo "Diseñar el servicio con previsión de que pueda estar disponible 24h al día durante 7 días a la semana" **no se evalúa**, puesto que depende del servicio en la nube AWS donde se aloja el código.

Se puede comprobar en la figura 6.8 que, ante los mismos textos de entrada, el BOT responde de formas diferentes, es decir, no devuelve siempre la misma respuesta al usuario.

Figura 6.8: Diferentes respuestas ante los mismos textos de entrada

Por tanto, **se ha cumplido el objetivo** "Crear un sistema de respuestas en lenguaje natural lo suficientemente variado para que el usuario sienta que tiene detrás una entidad no robótica".

6.4. Satisfacción de usuarios

Para evaluar la satisfacción de los usuarios con el producto, se formó un grupo de usuarios. Se pidió participar a 16 personas mayores, 9 mujeres y 7 hombres, a través de sus acompañantes. Ningún acompañante rechazó participar. 6 personas mayores rechazaron participar. Por tanto, el grupo quedó formado por 10 personas mayores y 9 cuidadores (un cuidador estaba a cargo de dos personas mayores; su padre y su madre).

Los motivos que manifiestan las personas mayores que recahzaron participar son:

- No creen que el BOT les vaya a ayudar.
- El BOT les causa rechazo.

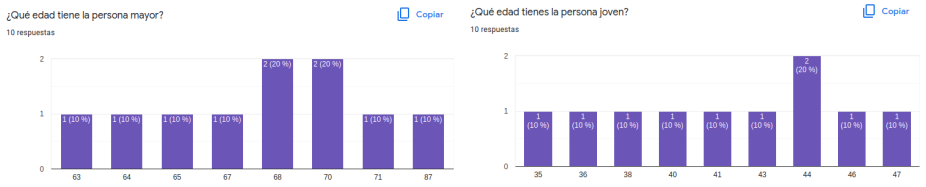
- Problemas de visión.

Previo al uso del BOT por los usuarios finales, se envía la encuesta inicial (ver anexo Sección A.3) a los 10 pares de usuarios. En esta encuesta, se pueden conocer datos de los participantes. Un 70 % de las personas mayores son mujeres, frente al 30 % de hombres (ver 6.9).

Figura 6.9: Sexo de los participantes

En la figura 6.10 se muestran las edades de los participantes. En 6.10a se refleja la edad de las peronas mayores, cuya media es de 69.3 años. En 6.10b se refleja la edad de las personas jóvenes, cuya media es de 41.4 años.

Respecto a las aplicaciones de mensajería utilizadas, se aprecia (ver 6.11) que la más usada, con el 100 % de uso, es WhatsApp, frente al 40 % que usa Telegram. Por tanto, el 60 % de usuarios tuvieron la necesidad de instalar Telegram.



(a) Edad de las personas mayores (b) Edad de las personas jóvenes

Figura 6.10: Edad de las personas mayores y de las jóvenes

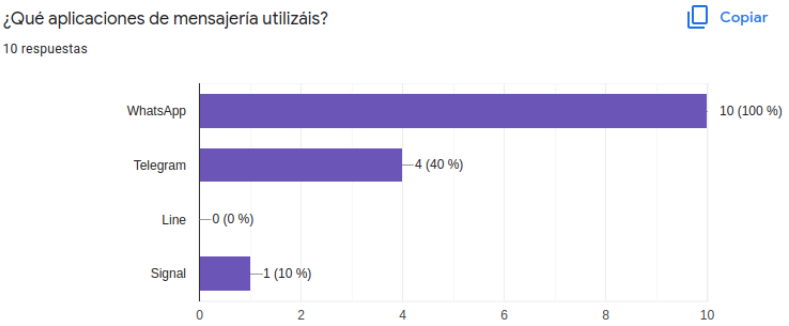


Figura 6.11: Apliaciones de mensajería utilizadas

Una vez desarrollada la funcionalidad completa del BOT, se les vuelve a encuestar (ver anexo Sección A.4) para que respondan a las mismas preguntas de la encuesta inicial, pero esta vez en pasado.

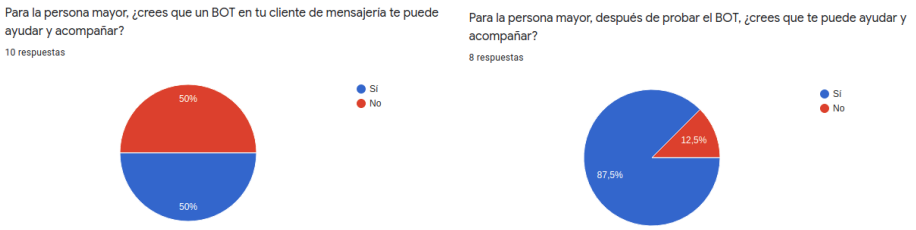
Por causas de fuerza mayor, un pares de persona mayor y acompañante no pudieron terminar, y un acompañante no pudo ayudar a la persona mayor, quien decidió abandonar el proyecto, por lo que hay 8 encuestas finales.

Se realiza una comparativa de cada pregunta, antes y después de probar el BOT.

En la figura 6.12 se aprecia la opinión de las personas mayores respecto a si un BOT en un cliente de mensajería puede acompañar y ayudar. La opinión mejora de un 50 % en encuesta inicial (6.12a) a un 87.5 % en la encuesta final (6.12b).

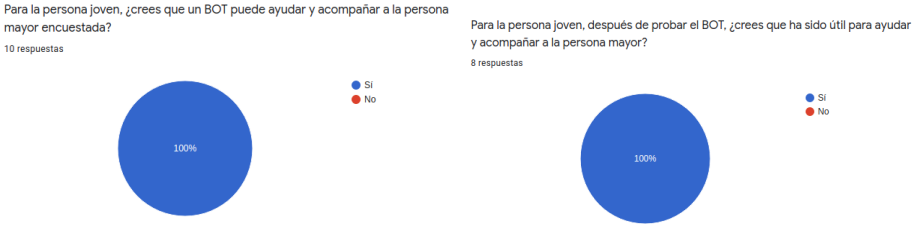
En la figura 6.13 se muestra la opinión de las personas jóvenes respecto a si un BOT en un cliente de mensajería puede acompañar y ayudar a personas mayores. La opinión se mantiene en el 100 % en las encuestas inicial (6.13a) y final (6.13b).

En la figura 6.14 se aprecia la opinión de los usuarios respecto a si les gustaría que el BOT tuviese iniciativa propia para escribir sin que se le pregunte nada. La opinión mejora del 70 % en la encuesta inicial (6.14a) al 100 % en la encuesta final (6.14b).



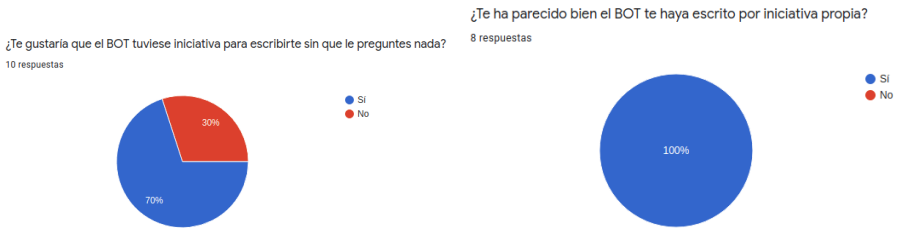
(a) Encuesta inicial (b) Encuesta final

Figura 6.12: Encuesta a personas mayores: acompañar y ayudar



(a) Encuesta inicial (b) Encuesta final

Figura 6.13: Encuesta a acompañantes: acompañar y ayudar



(a) Encuesta inicial (b) Encuesta final

Figura 6.14: ¿Te gustaría que el BOT tuviese iniciativa propia?

En la figura 6.15 se aprecia la opinión de los usuarios respecto a si les ayudaría que el BOT conociese su estado de ánimo. La puntuación tiene una media de 3.1 en la encuesta inicial (6.15a) frente al 3.9 en la encuesta final (6.15b), por lo que la opinión ha mejorado tras el uso del BOT.

En la figura 6.16 se aprecia la opinión de los usuarios respecto a si les ayudaría que el BOT les recordase la toma de medicamentos. La puntuación tiene una media de 4.1 en la encuesta inicial (6.16a) frente al 4.75 en la encuesta final (6.16b), por lo que la opinión ha mejorado tras el uso del BOT.

En la figura 6.17 se aprecia la opinión de los usuarios respecto a si les ayudaría que el BOT les recordase citas médicas. La puntuación tiene una media de 4.5 en la encuesta

(a) Encuesta inicial (b) Encuesta final

Figura 6.15: Conocer tu estado de ánimo

(a) Encuesta inicial (b) Encuesta final

Figura 6.16: Recordar medicamentos

(a) Encuesta inicial (b) Encuesta final

Figura 6.17: Recordar citas médicas

inicial (6.17a) frente al 4.75 en la encuesta final (6.17b), por lo que la opinión ha mejorado tras el uso del BOT.

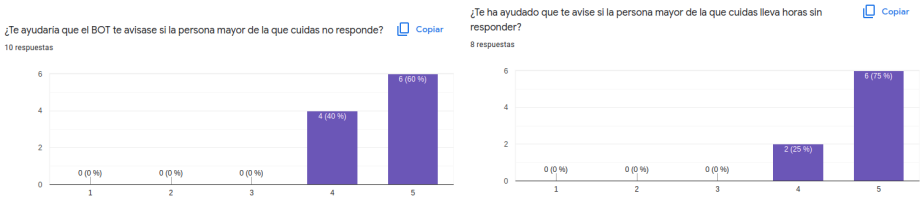
En la figura 6.18 se aprecia la opinión de los usuarios respecto a si les ayudaría que el BOT les preguntase de vez en cuando cómo se sienten. La puntuación tiene una media de 3.1 en la encuesta inicial (6.18a) frente al 4.25 en la encuesta final (6.18b), por lo que la opinión ha mejorado tras el uso del BOT.

En la figura 6.19 se aprecia la opinión de los cuidadores respecto a si les ayudaría que el BOT les avisase si la persona mayor no responde. La puntuación tiene una media de 4.6 en la encuesta inicial (6.19a) frente al 4.75 en la encuesta final (6.19b), por lo que la opinión ha mejorado tras el uso del BOT.

En la figura 6.20 se aprecia la opinión de los cuidadores respecto a si les ayudaría que

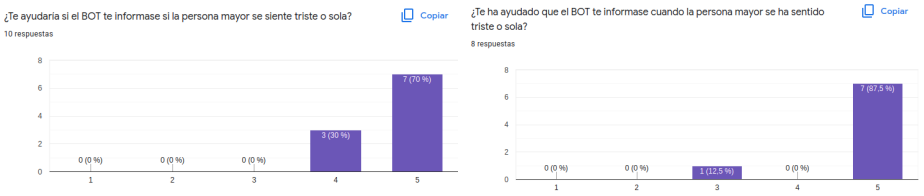
(a) Encuesta inicial (b) Encuesta final

Figura 6.18: Preguntar de vez en cuando cómo te sientes



(a) Encuesta inicial (b) Encuesta final

Figura 6.19: Aviso de inactividad de la persona mayor



(a) Encuesta inicial (b) Encuesta final

Figura 6.20: Aviso de que la persona mayor está triste o sola

el BOT les avisase si la persona mayor se siente triste o sola. La puntuación tiene una media de 4.7 en la encuesta inicial (6.20a) frente al 4.75 en la encuesta final (6.20b), por lo que la opinión ha mejorado ligeramente tras el uso del BOT.

La encuesta inicial preguntaba a los usuarios si les gustaría poder contar anécdotas al BOT, que las recordara y hablase sobre ellas. Esta funcionalidad (ver 6.21) tuvo una puntuación media de 3. Fue la funcionalidad con una puntuación media más baja, por lo que quedó fuera del proyecto su implementación, razón por la cual no se preguntó sobre ella en la encuesta final.

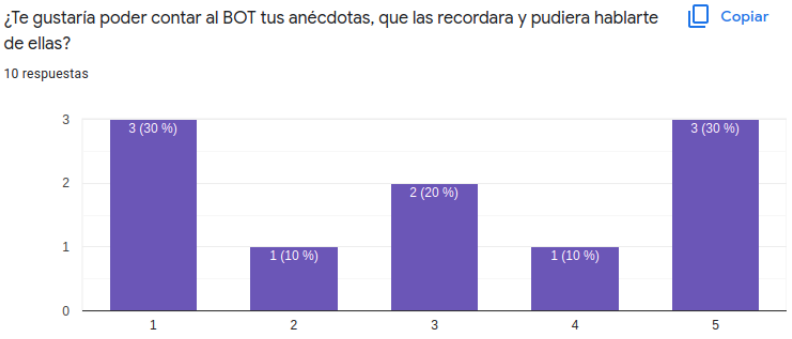


Figura 6.21: Contar anécdotas al BOT

En la encuesta final se les pregunta a los usuarios por su grado de satisfacción final

con el BOT. En la figura ver 6.22 se pueden ver los resultados.

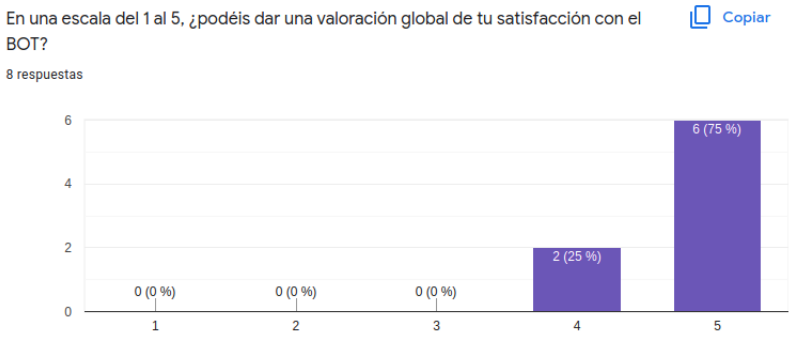


Figura 6.22: Grado de satisfacción global

Si se calcula la media de grado de satisfacción global de los usuarios con el BOT, se obtiene que ésta es de 4.75, por lo que **se ha superado el objetivo** "Evaluar la utilidad del BOT para las personas mayores y sus acompañantes, con una valoración global de satisfacción superior a 4.5 sobre 5".

A la vista de los resultados, se puede concluir que las personas mayores han sido capaces de usar el BOT de forma autónoma, pese a la ayuda inicial y necesaria de sus cuidadores, **se considera cumplido el objetivo** "Centrar la usabilidad y accesibilidad hacia un público objetivo de personas mayores para que sean capaces de usarlo de forma autónoma".

6.5. Desarrollo de un BOT conversacional

Una vez evaluados y superados los objetivos específicos, resta evaluar la consecución del objetivo final, "El objetivo principal del trabajo consiste desarrollar un BOT conversacional que sea capaz de averiguar la intención del usuario en base al texto de entrada con una tasa de acierto superior al 85 %."

Este objetivo **se considera cumplido** al alcanzar los objetivos específicos, puesto que ya se ha verificado que se ha desarrollado un BOT de calidad (Sección 6.3), capaz de identificar un 89.66 % de veces la intención de los usuarios (Sección 6.1), satisfaciendo sus necesidades, con una puntuación de 4.75 sobre 5 (Sección 6.4) en la encuesta final a los usuarios.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

Si se hace un repaso de los objetivos y la evaluación final de cada uno de ellos, podemos concluir que se han cumplido los objetivos fijados, tanto el objetivo principal como los específicos.

Se ha logrado elaborar un conjunto de datos a partir de una base de datos de diálogo. Sobre este conjunto se han detectado 25 necesidades reales de las personas mayores. Se ha podido entrenar un modelo con una tasa de acierto de 89.66 %, superior al 85 % establecido en el objetivo, en la tarea de detección de la intención. Se comprueba que, cuando el BOT es capaz de identificar una necesidad, pero no sabe reaccionar, se lo indica al usuario. Del mismo modo, cuando no entiende la intención del usuario, también se lo hace saber.

Para alcanzar los objetivos, se ha desarrollado un BOT para Telegram, que transmita las intenciones de los usuarios a una función Lambda en AWS a través de un API Gateway que hace de WebHook para Telegram. Esa función Lambda contiene el código en Python de un BOT conversacional, capaz de utilizar uno o varios modelos del lenguaje en sus diferentes componentes, para detectar la intención, activar comandos en base a la intención detectada, y emitir una respuesta final al usuario.

En el Estado del Arte se ha realizado una investigación de diferentes modelos del lenguaje. Para el desarrollo del BOT, se ha empleado Dialogflow, modelo creado por Google basado en el modelo del lenguaje BERT, uno de los modelos del lenguaje más potentes en la actualidad. A este modelo se le ha hecho un ajuste fino a partir de un conjunto de datos de diálogos de usuario, creado a partir de las interacciones de los usuarios con propio BOT. Para ello, se han generado diferentes conjuntos de datos, que se han etiquetado

manualmente y entrenado en la plataforma Dialogflow. Se ha descartado el uso de GPT-3 al ser una plataforma de pago. No obstante, se ha desarrollado una prueba de concepto con buenos resultados.

Para crear el BOT, se ha desarrollado en Python una arquitectura limpia de puertos y adaptadores, donde se definen en el dominio las abstacciones, implementándose éstas en la infraestructura. De este modo, se ha conseguido que el dominio no conozca los diferentes modelos del lenguaje ni las interfaces que lo van a usar. Gracias a este bajo acomplamiento, es posible cambiar de modelo de lenguaje fácilmente, al igual que crear nuevas interfaces para otros clientes de mensajería, como WhatsApp, el más utilizado según las encuestas.

Se ha verificado la calidad del software mediante pruebas automáticas, alcanzando una cobertura de código del 95 %, superior al 90 % requerido. También se han controlado excepciones para que un fallo del sistema no afecte a los usuarios finales y el BOT siempre les responda, informando del fallo ocurrido.

Para lograr que el BOT tenga iniciativa, se ha configurado un sistema de eventos en AWS que, cada 10 minutos, dispara una función Lambda que comprueba si tiene que recordar al usuario una cita médica o la toma de un medicamento. En la evaluación se ha verificado su correcto funcionamiento.

Se ha comprobado que el BOT ofrece respuestas variadas al usuario ante la misma pregunta, para lograr así una mejor experiencia de usuario.

Se ha contado con la participación de un grupo de usuarios, que han interactuado con el BOT, alimentando la base de datos de diálogos, que ha permitido crear el conjunto de datos sobre el que se han detectado sus necesidades y que ha servido para el entrenamiento del modelo.

A estos usuarios se les ha enviado una encuesta final para comprobar que no sólo se han cumplido los objetivos en cuanto al desarrollo software, sino que el producto final es también de utilidad para estos usuarios, obteniendo una nota de satisfacción global de 4.75 sobre 5.

Esta encuesta ha servido para validar la utilidad del desarrollo de de una herramienta tipo BOT que ha cubierto las necesidades de los usuarios.

Se ha verificado que se han cumplido los objetivos específicos y el objetivo general, al haber desarrollado un BOT, con calidad software para responder ante errores, capaz de detectar la intención del usuario en un 89.66 % de ocasiones, que es útil para los usuarios finales.

7.2. Líneas de trabajo futuro

En este trabajo se ha desarrollado producto basado en un BOT conversacional capaz de ayudar y acompañar a personas mayores. A partir de esta base sólida construida, el crecimiento del proyecto puede venir por varias vías:

7.2.1. Mejoras en la interacción humano-máquina

En base a las encuestas, se ha comprobado que Telegram no es la primera opción de mensajería utilizada por los usuarios. Por ello, y gracias al diseño arquitectónico escalable, se pueden desarrollar nuevos componentes para diferentes plataformas desde las que el usuario pueda conversar con el BOT, como podrían ser:

- Nuevos clientes de mensajería, como WhatsApp, el más utilizado según las encuestas hechas a usuarios.
- Una página web con un chatbot integrado.
- Un robot físico, capaz de reconocer voz y de hablar con el usuario.
- Altavoces de Google y Alexa, creando aplicaciones para sus asistentes que interactúen con el BOT.

Uno de los motivos de rechazo por parte de las personas mayores a la hora de participar en en el proyecto, fue la dificultad de interactuar con un BOT escribiendo y leyendo mensajes de texto. Telegram tiene una opción para que el usuario pueda dictar por voz al BOT, pero aun así, a los usuarios le resultó complicado de utilizar, y, en cualquier caso, tenían que leer en texto la respuesta del BOT.

Por tanto, una posible mejora, es utilizar un sistema STT y TTS, ya sea a través de APIs de terceros, como la de Google, o elaborando uno propio. Como se ha comentado, este sistema podría estar embebido en un robot físico, o directamente hacerlo compatible con los asistentes de Google o Alexa.

Por otro lado, en las interacciones de los usuarios con el BOT, se ha comprobado que, en muchas ocasiones, se dirigen en segunda persona al BOT, como si se tratase de una persona. Algunos cuidadores comentaron que su madre “pensaba que había detrás una persona” y que tenían que explicarle que era un BOT que “acababa de nacer” y estaba

aprendiendo según se hablaba con él. Este hecho resalta la necesidad de dotar al BOT de una personalidad para que la interacción con el usuario sea más humana.

En esa misma línea, los usuarios también demandan la necesidad de que el BOT sea capaz de seguir una conversación en lenguaje natural de un tema cotidiano, entendiendo el contexto de la conversación.

Finalmente, el BOT actualmente está dirigido a personas mayores, pero como futura mejora, se podría ofrecer este BOT a otro tipo de personas con necesidades especiales, captarlas a través del BOT y ofrecerles ayuda en lo que este público objetivo necesitase.

7.2.2. Mejoras en el modelo entrenado

Para entrenar un modelo, se ha necesitado elaborar un conjunto de datos etiquetado, y entrenar un modelo ya pre-entrenado en base a ese conjunto de datos creado. Por tanto, las mejoras en este proceso pasan por mejorar la forma de crear ese conjunto de datos y los modelos del lenguaje a emplear.

Para facilitar la creación del conjunto de datos, es posible mejorar el proceso de etiquetado manual, utilizando herramientas terceras, como Rubrix, que ofrece una interfaz web para la tarea de etiquetado manual y análisis del conjunto de datos.

Respecto a los modelos de lenguaje a emplear, se pueden crear diferentes líneas de investigación con diferentes modelos, para mejorar la conversación natural. Estos modelos podrían ser MarIA, o GPT-3, ya probado y descartado por ser de pago. Se puede barajar la posibilidad de entrenar un modelo propio desde cero.

Gracias a la arquitectura diseñada, la inclusión de estos nuevos modelos del lenguaje no tendría el mayor impacto en el código existente, y, una vez cerrada una línea de investigación con un resultado positivo, sólo habría que incluir un nuevo componete en el BOT.

Otra línea de investigación podría nacer para probar Amazon Lex, ya que, al estar integrado con AWS, podría facilitar su integración con el BOT, al estar desplegado en AWS.

7.2.3. Extensión de la funcionalidad

Gracias a las encuestas a los usuarios, se han detectado diversas necesidades que pueden tener las personas mayores y que se podrían llegar a transformar en nuevas apliciones a integrar, que se sumasen a las actuales aplicaciones médica, para recordar citas y

medicamentos, de estado de ánimo o de detección de inactividad. Algunas de estas nuevas aplicaciones sugeridas por los usuarios son:

- **Conexión IoT:** para poder pedir al BOT que encienda o apague luces, o conectarlo a diferentes sensores que puedan informar al cuidador de caídas en el baño o, mediante una pulsera, de un elevado ritmo cardíaco.
- **Nutricionista:** para relacionar alimentos con enfermedades y medicamentos, y así aconsejar qué alimentos son más adecuados para el usuario.
- **Juegos:** para tener despierta la mente, el BOT puede plantear al usuario preguntas sobre anécdotas que le cuente el usuario y detectar si éste ha acertado. Esto puede servir para un futuro diagnóstico temprano de enfermedades, como Alzheimer.
- **Lista de la compra:** para añadir productos a una lista de la compra, como recetas médicas.
- **Meteorología:** para informar del tiempo, dar consejos de salud ante olas de calor o diferentes adversidades climatológicas.
- **Noticias:** para informar al usuario de noticias que sean de su interés.

Bibliografía

AAL Home 2020 - AAL Programme. (s.f.). http://www.aal-europe.eu/

Alonso Martín, F., Castro-González, Á., Gorostiza, J. & Salichs, M. (2015). Augmented Robotics Dialog System for Enhancing HumanRobot Interaction. *Sensors (Basel, Switzerland)*, *15*, 15799-15829. https://doi.org/10.3390/s150715799

Amazon. (s.f.). *Amazon Lex*. https://aws.amazon.com/es/lex/

Barbero, Á. & Vaca, A. (2022). *Webinar (AI Tech Talk): Estado del arte en modelos de lenguaje en español - YouTube*. https://www.youtube.com/watch?v=m3OcuAf5l9I

Bentivoglio, M. & Zucconi, G. G. (2018). *Cuando el cerebro envejece. Mitos y certezas sobre un proceso universal (e inevitable)* (S. S.L., Ed.).

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, *2020-December*. https://doi.org/10.48550/arxiv.2005.14165

Carlos, J. & Gualdrón, A. (2018). Robots para el cuidado de personas mayores. Taxonomía de una promesa, *24*, 43-60. https://doi.org/10.14201/aula2018244360

Conde-Ruiz, J. I. (2021). El proceso de envejecimiento en España.

Dale, R. (2021). GPT-3: What's it good for? *Natural Language Engineering*, *27*, 113-118. https://doi.org/10.1017/S1351324920000601

Devlin, J. (2018). *BERT*. https://github.com/google-research/bert

Doblas, J. L. & Conde, M. D. P. D. (2018). El sentimiento de soledad en la vejez. *Revista Internacional de Sociologia*, *76*. https://doi.org/10.3989/RIS.2018.76.1.16.164

Eisenstein, J. (2018). Natural Language Processing.

Google. (s.f.). *Dialogflow Google Cloud*. https://cloud.google.com/dialogflow?hl=es

Gregorio, P. G. (2018). *Neurodegeneración, Alzheimer, Parkinson y ELA*.

Gutiérrez, A. (2022). *PlanTL-GOB-ES/lm-spanish: Official source for spanish Language Models and resources made @ BSC-TEMU within the "Plan de las Tecnologías del Lenguaje"(Plan-TL)*. https://github.com/PlanTL-GOB-ES/lm-spanish

Gutiérrez-Fandiño, A., Armengol-Estapé, J., Pàmies, M., Llop-Palao, J., Silveira-Ocampo, J., Carrino, C. P., Armentano-Oller, C., Rodriguez-Penagos, C., Gonzalez-Agirre, A. & Villegas, M. (2022). MarIA: Spanish Language Models MarIA: Modelos del Lenguaje en Español. https://doi.org/10.26342/2022-68-3

HuggingFace. (s.f.). *Hugging Face*. https://huggingface.co/

IBM. (s.f.). *IBM Watson*. https://www.ibm.com/es-es/products/watson-assistant

Kim, S. (2021). Exploring How Older Adults Use a Smart Speaker-Based Voice Assistant in Their First Interactions: Qualitative Study. *JMIR mHealth and uHealth*, *9*, e20427. https://doi.org/10.2196/20427

los Santos Cicutto, S. D. (2017). Experiencia de uso de asistentes de voz sin GUI en personas mayores.

Martin, R. C. (2018). Arquitectura limpia. Guía para especialistas en la estructura y el diseño software.

Pradhan, A., Findlater, L. & Lazar, A. (2019). "Phantom Friend.ºr "Just a Box with Information": Personification and Ontological Categorization of Smart Speaker-based Voice Assistants by Older Adults, *3*. https://doi.org/10.1145/3359316

Rodríguez, V. G. (2020). Sistema conversacional de ayuda a personas mayores basado en Dialogflow. https://uvadoc.uva.es/handle/10324/41135

Romero, M., Casadevante, C. & Montoro, H. (2020). How to create a psychologist-chatbot. *Papeles del Psicologo*, *41*, 27-34. https://doi.org/10.23923/pap.psicol2020.2920

Rosa, J. D. L., Ponferrada, E. G., Villegas, P., Salas, P. G. D. P., Romero, M., Grandury, M. & Project, B. (2022). BERTIN: Efficient Pre-Training of a Spanish Language Model using Perplexity Sampling. https://doi.org/10.26342/2022-68-1

Sánchez, P. A. M. (2014). Entornos de ayuda a la vida independiente de personas mayores sobre servicios de redes de próxima generación.

Schwaber, K. & Sutherland, J. (2020). Manifesto for Agile Software Development.

Vajjala, S., Majumder, B., Gupta, A. & Surana, H. (2020). *Practical Natural Language Processing A Comprehensive Guide to Building Real-World NLP Systems*.

Vásquez, M. A. C., Huerta, M. H. V., Jaime, L. & Quispe, P. (2009). *Procesamiento de lenguaje natural*.

Vaswani, A., Brain, G., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017). Attention Is All You Need.

Vicente, M., Barros, C., Peregrino, F. S. & Agu, F. (2021). La generación de lenguaje natural: análisis del estado actual. https://doi.org/10.13053/CyS-19-4-2196

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P. V., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., ... Rush, A. M. (2020). *Transformers: State-of-the-Art Natural Language Processing*. https://github.com/huggingface/

Apéndice A

Anexos

A.1. Artículo científico

BOT conversacional para acompañar y cuidar a personas mayores

Diego de los Reyes Rodríguez

Universidad Internacional de la Rioja, Logroño (España)

30 de junio de 2022

RESUMEN

Las personas pierden progresivamente su autonomía a medida que envejecen. Esto hace necesario que cuenten con ayuda de terceros. Se desarrolla un BOT para Telegram usando la plataforma Dialogflow en español para ayudar a los usuarios, acompañándoles y cuidándoles con acciones como recordarles su medicación o avisar a sus cuidadores de su estado real. El objetivo principal es lograr que el BOT sea capaz de averiguar la intención del usuario en base al texto de entrada con una tasa de acierto superior al 85 %. El BOT se les proporcionó a diez personas mayores con sus jóvenes cuidadores durante el desarrollo del producto. El modelo del lenguaje ajustado en Dialogflow con el conjunto de datos creado con sus interacciones con el BOT, alcanzó una tasa de acierto del 89.66 % en la tarea de detección de la intención. El grado de satisfacción final de los usuarios ha sido de 4.75 sobre 5.

PALABRAS CLAVE

BOT conversacional, cuidadores, personas mayores, procesamiento del lenguaje natural, modelo del lenguaje ajustado

I. INTRODUCCIÓN

¿Se pueden identificar las intenciones de los mayores con un BOT que les acompañe y cuide sin que les cause rechazo? Este trabajo se centra en ayudarles a sentirse más autónomos y menos dependientes.

Al envejecer, se pierden capacidades [1] de forma irreversible [2], con consecuencias emocionales y sociales, manifestadas en emociones negativas, como la soledad [3].

En 2018, un 18,9 % de la Unión Europea (UE) tenía 65 años o más [4]. La iniciativa Active and Assisted Living (AAL) [5], pronostica para 2070 que más de la mitad de la población de la UE será mayor de 65 años. España está envejeciendo [6]. La tasa de dependencia (ver 1) ha crecido en los últimos años. Según estadísticas del INE, AIReF y Eurostat, el aumento de población mayor va a continuar hasta 2050, en el que puede llegar a doblarse (ver 2), incrementando la demanda de servicios sociosanitarios que mantengan la autonomía y calidad de vida de los mayores [7]. En 2050 España será uno de los países más envejecidos.

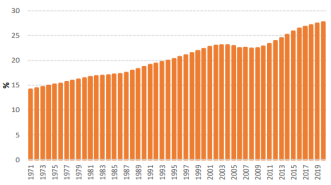


Figura 1: Tasa de dependencia a los 67 años (1971 a 2020) [6]

Soluciones como los asistentes de Google y Alexa o robots asistentes no son por ahora definitivas para para acompañar y cuidar a cada persona mayor. La población mayor va en aumento y no se puede revertir el envejecimiento. Este trabajo se centra mejorar la autonomía de

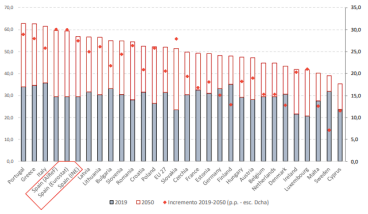


Figura 2: Tasa de dependencia a los 65 años. Año 2050 [6]

los mayores con un BOT para Telegram, desarrollado en Python y desplegado en *Amazon Web Services* (AWS). Se cuenta con un grupo de usuarios de mayores y cuidadores que interactúan con el BOT y opinan con encuestas sobre su utilidad. En base a sus interacciones se elabora un conjunto de datos para entrenar diferentes modelos, que se usan en un BOT con arquitectura escalable con iniciativa para iniciar una conversación y ayudar o acompañar a una persona mayor o informar a su cuidador de su estado, sin sustituirle.

Una motivación social y un uso responsable de la tecnología incitan este trabajo, encuadrado en el Procesamiento del Lenguaje Natural (PLN), área de la Inteligencia Artificial (IA) que hace accesible el lenguaje humano a las máquinas, para lo cual, debe interpretar las oraciones proporcionadas por un humano durante un proceso de comunicación [8] [9]. El objetivo es lograr que el BOT identifique la intención del usuario con una tasa de acierto superior al 85 %.

II. ESTADO DEL ARTE

PLN resuelve diversas tareas, como clasificación de textos, extracción de información o generación de lenguaje natural [10][9] [11] [12] [8], con diferentes técnicas que han evolucionado hasta llegar a generar modelos del lenguaje con redes neuronales profundas (DNN). Entre sus aplicaciones están los BOTs conversacionales, que utilizan modelos del lenguaje para resolver tareas de identificación de intenciones o generación de respuestas en lenguaje natural. Se invocan desde diferentes clientes, como Telegram.

En 2017 Google presenta Transformers [13]. En la figura 3 se muestra su arquitectura.

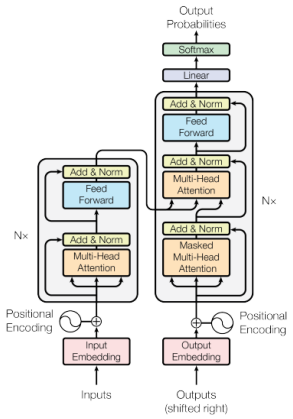


Figura 3: El Transformer - arquitectura. [13]

Los transformers dieron lugar a nuevos modelos [11], reflejados en la tabla 1.

Modelo	Fecha
BERT	2018
RoBERTa	2019
GPT / GPT-2	2019
DistilBERT	2019
XLM/RoBERTa	2019
GPT-3	2020
MarIA	2022
Google PaLM	2022

Cuadro 1: Modelos del lenguaje con DNN [11]

En [14] se repasan los modelos en español, como BETO, a finales de 2019, BERTIN [15], del año 2022, y MarIA [16], presentado en marzo de 2022 y disponible en [17].

Existen plataformas para crear BOTs o entrenar modelos en la nube, como Google Dialogflow [18], que utiliza BERT, IBM Watson [19], Amazon Lex [20] o Hugging Face [21].

Para evaluar un modelo se mide cómo se comporta ante nuevos datos [10], de forma intrínseca, calculando la tasa de acierto, precisión, *recall* o *F1 score*, o extrínseca, con métricas basadas en las reglas del negocio.

Un BOT conversacional es un sistema que permite a los usuarios interactuar con él mediante lenguaje natural [10]. Su origen [22] se remonta a 1966, con el ELIZA, primer chatbot de la historia. En 1995 se desarrolla Alicebot, basado en ELIZA. Hoy podemos encontrar a *Google Assistant* (Google), *Siri* (Apple) o *Cortana* (Microsoft).

La arquitectura de un sistema conversacional [10] (ver 4) tiene componentes de reconocimiento automático del lenguaje (transforma lenguaje hablado en escrito), comprensión del lenguaje (identifica la intención y extrae la semántica), gestión del diálogo (mantiene el estado y flujo de conversación), generación de respuesta (transmite la respuesta al usuario) y síntesis de voz (convierte el texto a habla).

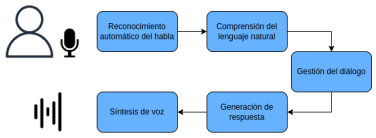


Figura 4: Flujo de agentes conversacionales (adaptado de [10])

El programa AAL [5] financia la innovación y desarrollo de productos y servicios para mayores y sus cuidadores. En [7] se realiza una ontología para modelar servicios AAL que ofrecen soporte a su movilidad e independencia.

Existen robots [4] para el cuidado de mayores que realizan tareas médicas, domésticas, sociales (como Maggie [23]), recreativas, educativas, rehabilitadoras o terapéuticas (ver 5).

El estudio de [24] sobre la experiencia de uso de asistentes de voz en mayores de 60 años concluye que su uso puede mejorar la comunicación con los asistentes al resultarles más sencillo e intuitivo. En el estudio de [25], se instalan altavoces inteligentes en sus hogares durante tres semanas para estudiar su interacción. Se les entrevista a diario. Se concluye remarcando

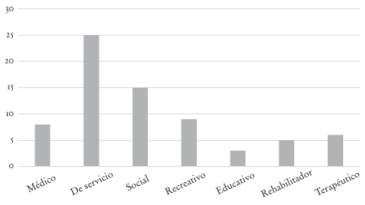


Figura 5: Tipos de robots encontrados [4]

el potencial de esta tecnología en interacciones sociales. En [26] se analiza cómo los mayores perciben el beneficio de los altavoces.

El trabajo [27] presenta un agente conversacional para ayudar a los mayores a combatir la soledad, con *Dialogflow* de *Google*. Ofrece funciones como adivinanzas, chistes, juegos o pedir cita al médico. Algunas no se pueden realizar completamente por voz. Se realizan encuestas a los usuarios para medir la calidad del agente, que recibió buenas opiniones.

III. OBJETIVOS Y METODOLOGÍA

El objetivo principal es desarrollar un BOT capaz de averiguar la intención del usuario con una tasa de acierto superior al 85 %.

Se establecen los siguientes objetivos específicos:

- Elaborar un conjunto de datos para entrenar un modelo con una tasa de acierto superior al 85 % detectando la intención.
- Descubrir necesidades reales de las personas mayores y clasificarlas en intenciones según el diálogo.
- Investigar modelos preentrenados y plataformas de PLN para entrenar al BOT.
- Desarrollar un BOT con el modelo entrenado que responda a la intención identificada.
- Diseñar una arquitectura de un BOT capaz de hablar de forma proactiva ante determinados eventos y sea escalable.

- Diseñar el servicio con previsión de que pueda estar disponible 24h al día durante 7 días a la semana.
- Evaluar la utilidad del BOT por los usuarios con una valoración global de satisfacción superior a 4.5 sobre 5.
- Dotar al sistema de robustez para recuperarse de errores sin que el usuario final se vea afectado.
- Centrar la usabilidad y accesibilidad hacia un público objetivo de personas mayores.
- Crear un sistema de respuestas variado en lenguaje natural.

Como metodología, se escogen algunos aspectos del marco de trabajo Scrum [28] y se utiliza un tablero Kanban para la gestión de tareas.

IV. CONTRIBUCIÓN

El diseño del BOT sigue los principios de arquitecturas limpias [29]. Se consigue un código escalabe y preparado para pruebas automáticas.

Se puede acceder al código fuente en:
https://github.com/diegorys/bot-tfm/releases/tag/1.0.0-tfm

El servicio se despliega en AWS (ver 6).

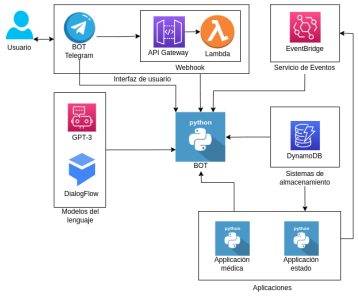


Figura 6: Infraestructura en la nube.

El dominio es un paquete al que invoca una interfaz, procesa la orden y emite una respuesta a quien le ha invocado. El diseño se muestra en la figura 7.

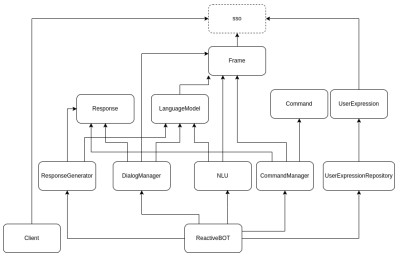


Figura 7: Dominio del BOT

La interfaz "LanguageModel" especifica los métodos a implementar por los modelos del lenguaje para resolver tareas de PLN. Este proyecto utiliza el modelo de lenguaje Dialogflow.

El componente "CommandManager" ejecuta, si existe, el comando asociado a la intención.

Un sistema de eventos se conecta al cliente, que actúa como BOT proactivo, para avisar al usuario de un evento.

Se realizan pruebas automáticas con *pytest* de Python que verifican el funcionamiento del BOT.

V. RESULTADOS O EVALUACIÓN

En el conjunto de datos de 236 diálogos se han identificado 25 necesidades.

Con el 80 % del conjunto de datos para entrenar en Dialogflow y el 20 % para validación,el modelo alcanza el **89.66 % de tasa de acierto** al identificar la intención del usuario.

El desarrollo del BOT es accesible desde **@CuidadorMayoresBot**.

En la figura 8 se observa un diálogo donde el BOT entiende la intención y sabe reaccionar, identifica la intención pero no sabe actuar, y no es capaz de identificar la necesidad.

En el diálogo de la figura 9 responde al saludo, registro de medicación, cita médica, estado de ánimo y despedida.

En la figura 10 informa ante un error técnico y en la 11 de un usuario no reconocido.

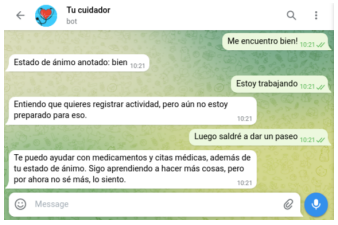


Figura 8: Diferentes situaciones

Figura 9: Intenciones identificadas

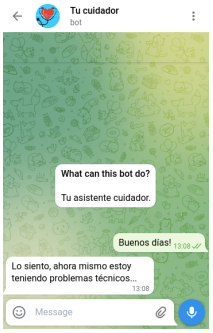


Figura 10: Error técnico no controlado

Se cubre un 95 % de código con 54 pruebas automáticas sin fallos (ver 12).

El BOT es capaz de iniciar conversación. Cuando cumple la hora indicada, con un margen de 15 minutos de antelación, recuerda al usuario lo que le ha pedido (ver 13).

Figura 11: Usuario no reconocido

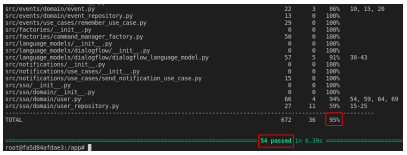


Figura 12: Resultado de ejecución de pruebas automáticas

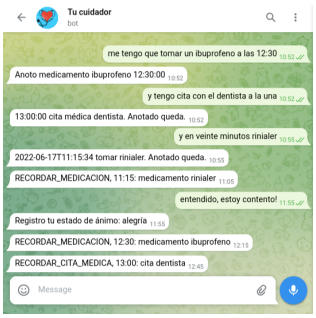


Figura 13: El BOT recuerda al usuario su medicación y citas médicas

La arquitectura software permite la creación de nuevos componentes software sin que afecte al dominio de la aplicación.

Ante los mismos textos de entrada, el BOT responde de formas diferentes (ver 14).

Se evalúa la satisfacción de los usuarios con una media de 4.74 al encuestar a 10 mayores y sus cuidadores (ver 15).

Figura 14: Diferentes respuestas

Figura 15: Grado de satisfacción global

VI. DISCUSIÓN O ANÁLISIS DE RESULTADOS

Se cumple el objetivo de descubir necesidades reales de los mayores al identificar 25 necesidades.

Se ha logrado elaborar un conjunto de datos y entrenar un modelo con una tasa de acierto superior al 85 % al detectar la intención con un 89.66 % de acierto.

Se ha cumplido el objetivo de investigar diferentes modelos del lenguaje y plataformas en el estado del arte y se ha usado Dialogflow.

Se ha interactuado con el BOT para verificar que se ha desarrollado un BOT conversacional que identifique la intención del usuario usando un modelo del lenguaje y actúe en consecuencia.

El objetivo de diseñar una arquitectura de un BOT escalable que hable de forma proactiva se ha cumplido con el diseño de una arquitectura limpia.

El objetivo de dar robustez al sistema se ha cumplido con un 95 % de cobertura de código con pruebas automátias sin fallos y el tratamiento de excepciones.

La disponibilidad del servicio no se evalúa y queda supeditada a la nube AWS.

En los resultados se ha visto cómo el BOT ofrece respuestas variadas, por lo que este objetivo ha quedado cubierto.

En las encuestas se ha alcanzado una media de 4.75 de satisfacción, por lo que se ha superado el objetivo de superar un 4.5 sobre 5.

Los mayores han podido de usar el BOT, por lo que considera cumplido el objetivo de centrar la usabilidad y accesibilidad en ellos.

El objetivo principal **se considera cumplido** al alcanzar los objetivos específicos. Se ha verificado que se ha desarrollado un BOT de calidad, que identifica un 89.66 % de veces la intención de los usuarios, y satisface sus necesidades, con una puntuación de 4.75 sobre 5.

VII. CONCLUSIONES

Podemos concluir que se ha cumplido el objetivo principal y los específicos.

Se ha elaborado un conjunto de datos y se han detectado 25 necesidades reales de los mayores. Se ha entrenado un modelo con una tasa de acierto de 89.66 %, superior al 85 % del objetivo al detectar la intención. Se comprueba que el BOT siempre responde, indicando que no entiende al usuario o que hay un error si procede.

Para alcanzar los objetivos, se ha desarrollado un BOT para Telegram, comunicado con una función Lambda en AWS disparada por un API Gateway invocado por Telegram. El BOT es capaz de utilizar uno o varios modelos del lenguaje.

Se ha realizado una investigación de diferentes modelos del lenguaje y plataformas y se ha elegido Dialogflow, con un ajuste fino en base al un conjunto de datos.

El BOT se ha desarrollado en Python con una arquitectura limpia donde el dominio no conoce los modelos del lenguaje ni las interfaces que lo usan. Este bajo acomplamiento permite cambiar de modelo de lenguaje y crear nuevas

interfaces para otros clientes de mensajería, como WhatsApp.

Se ha verificado la calidad del software mediante pruebas automáticas, con un 95 % de código cubierto. Se han controlado excepciones para que el BOT responda informando del fallo ocurrido.

Para que el BOT tenga iniciativa, se ha configurado un sistema de eventos en AWS que, cada 10 minutos, dispara una función Lambda que comprueba si tiene que recordar al usuario una cita médica o la toma de un medicamento. Se ha verificado su correcto funcionamiento.

Se ha comprobado que el BOT ofrece respuestas variadas ante la misma pregunta, mejorando la experiencia de usuario.

Se ha contado con un grupo de usuarios que ha alimentado la base de datos de diálogos para crear el conjunto de datos, detectar sus necesidades y entrenar el modelo. Se les ha enviado una encuesta de satisfacción, a la que han respondido puntuando con un 4.75 sobre 5.

Se han cumplido los objetivos específicos y el objetivo general, al haber desarrollado un BOT, con calidad para responder ante errores, que detecta la intención del usuario en un 89.66 % de ocasiones y es útil para los usuarios.

A. Líneas de trabajo futuro

Se pueden emplear diferentes plataformas para conversar con el BOT, como WhatsApp, una web con un chatbot integrado, un robot físico o los altavoces de Google o Alexa para interactuar por voz.

Se puede dotar al BOT de personalidad para humanizarlo y hacerle capaz de seguir una conversación en lenguaje natural de un tema cotidiano.

Se puede ofrecer este BOT a otro tipo de personas con otras necesidades especiales.

Para facilitar la creación del conjunto de datos se pueden usar herramientas como Rubrix, que ofrece una interfaz web para el etiquetado manual y análisis del conjunto de datos.

Se pueden crear diferentes líneas de investigación de modelos del lenguaje, como MarIA o GPT-3, o entrenar uno desde cero, para mejorar la conversación natural, o probar Amazon Lex, ya que, al estar integrado con AWS, podría facilitar su integración con el BOT desplegado en AWS.

Las diversas necesidades de los mayores sugeridas por ellos mismos en las encuestas se pueden transformar en aplicaciones, como conexión con IoT (dispositivos domóticos y pulseras de salud), relación entre medicamentos y nutrición, juegos para despertar la mente, permitir crear listas de compra, informar del tiempo o enviar noticias de interés.

Referencias

[1] Marina Bentivoglio and Gigliola Grassi Zucconi. *Cuando el cerebro envejece. Mitos y certezas sobre un proceso universal (e inevitable)*. 2018.

[2] Pedro Gil Gregorio. *Neurodegeneración, Alzheimer, Parkinson y ELA*. 2018.

[3] Juan López Doblas and Mariá Del Pilar DIáz Conde. El sentimiento de soledad en la vejez. *Revista Internacional de Sociologia*, 76, 3 2018.

[4] Juan Carlos and Aceros Gualdrón. Robots para el cuidado de personas mayores. taxonomía de una promesa. 24:43–60, 7 2018.

[5] Aal home 2020 - aal programme.

[6] José Ignacio Conde-Ruiz. El proceso de envejecimiento en españa. 2021.

[7] Pedro Antonio Moreno Sánchez. Entornos de ayuda a la vida independiente de personas mayores sobre servicios de redes de próxima generación. 2014.

[8] Jacob Eisenstein. Natural language processing. 2018.

[9] Mg Augusto Cortez Vásquez, Mg Hugo Vega Huerta, Lic Jaime, and Pariona Quispe. Procesamiento de lenguaje natural, 2009.

[10] Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, and Harshit Surana. Practical natural language processing a comprehensive guide to building real-world nlp systems, 2020.

[11] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick Von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M Rush. Transformers: State-of-the-art natural language processing, 2020.

[12] Marta Vicente, Cristina Barros, Fernando S Peregrino, and Francisco Agu. La generación de lenguaje natural: análisis del estado actual. 2021.

[13] Ashish Vaswani, Google Brain, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[14] Álvaro Barbero and Alejandro Vaca. Webinar (ai tech talk): Estado del arte en modelos de lenguaje en español - youtube, 2022.

[15] Javier De La Rosa, Eduardo G Ponferrada, Paulo Villegas, Pablo González De Prado Salas, Manu Romero, María Grandury, and Bertin Project. Bertin: Efficient pre-training of a spanish language model using perplexity sampling. 2022.

[16] Asier Gutiérrez-Fandiño, Jordi Armengol-Estapé, Marc Pàmies, Joan Llop-Palao, Joaquín Silveira-Ocampo, Casimiro Pio Carrino, Carme Armentano-Oller, Carlos Rodriguez-Penagos, Aitor Gonzalez-Agirre, and Marta Villegas. Maria: Spanish language models maria: Modelos del lenguaje en español. 2022.

[17] Asier Gutiérrez. Plantl-gob-es/lm-spanish: Official source for spanish language models and resources made @ bsc-temu within the "plan de las tecnologías del lenguaje"(plan-tl)., 2022.

[18] Google. Dialogflow google cloud.

[19] IBM. Ibm watson.

[20] Amazon. Amazon lex.

[21] HuggingFace. Hugging face.

[22] Miriam Romero, Cristina Casadevante, and Helena Montoro. How to create a psychologist-chatbot. *Papeles del Psicologo*, 41:27–34, 4 2020.

[23] Fernando Alonso Martín, Álvaro Castro-González, Javi Gorostiza, and Miguel Salichs. Augmented robotics dialog system for enhancing humanrobot interaction. *Sensors (Basel, Switzerland)*, 15:15799–15829, 07 2015.

[24] Santiago De los Santos Cicutto. Experiencia de uso de asistentes de voz sin gui en personas mayores. 2017.

[25] Alisha Pradhan, Leah Findlater, and Amanda Lazar. "phantom friend.ºr "just a box with information": Personification and ontological categorization of smart speaker-based voice assistants by older adults. 3, 2019.

[26] Sunyoung Kim. Exploring how older adults use a smart speaker-based voice assistant in their first interactions: Qualitative study. *JMIR mHealth and uHealth*, 9:e20427, 1 2021.

[27] Víctor Gobernado Rodríguez. Sistema conversacional de ayuda a personas mayores basado en dialogflow. 2020.

[28] Ken Schwaber and Jeff Sutherland. Manifesto for agile software development. 2020.

[29] R. C. Martin. Arquitectura limpia. guía para especialistas en la estructura y el diseño software. 2018.

A.2. Conjunto de datos y modelos con Jupyter Notebook

Análisis del conjunto de datos y evaluación de modelos

June 17, 2022

1 Conjuntos de datos

En este apartado, se describen los diferentes conjuntos de datos obtenidos en base a los diálogos. Cada registro está formado por una columna de entrada, que es el texto a clasificar, y una columna de salida, que es la intención en la que se clasifica el texto.

Según se define en el apartado **4. Identificación de requisitos** de la memoria, se van a tratar las siguientes intenciones:

- Registrar saludo
- Registrar toma de medicamento
- Registrar estado emocional
- Registrar cita médica
- Registrar despedida

Los objetivos cubiertos son:

- Elaborar un conjunto de datos que permita entrenar un modelo con una tasa de acierto superior al 85% en la tarea de detección de la intención: se extraen de la base de datos de diálogo las expresiones de usuario, se clasifican manualmente y se genera el conjunto de datos para entrenar en la plataforma Dialogflow. Se evalúa el modelo.
- Descubrir necesidades reales de las personas mayores a partir de la base de datos de diálogos y clasificarlas en intenciones según el diálogo: en base al conjunto de datos etiquetado, se listan las necesidades y el número de registros que se han clasificado en cada texto. De este modo, es posible evaluar qué necesidades son las más importantes.

[1]:
```
import json
import matplotlib.pyplot as plt
import pandas as pd

def getDataset(version, sufix=""):
    path = f"/home/jovyan/data/dataset/{version}/data{sufix}.json"
    with open(path, 'r') as f:
      data = json.load(f)
    return data

def getDatasetAsDictionaryText(version):
    dataset = getDataset(version)
    datasetDictionary = {}
```

```
    for item in dataset:
        datasetDictionary[item["text"]] = item
    return datasetDictionary

def dataToCSV(dataset, version, sufix=""):
    entries = []
    for entry in dataset:
        text = entry["text"]
        intent = entry["intent"]
        entryText = f"{text};{intent}"
        entries.append(entryText)
    path = f"/home/jovyan/data/dataset/{version}/data{sufix}.csv"
    with open(path,'w') as f:
        for line in entries:
            f.write(line)
            f.write('\n')

def writeCSV(entries, version, sufix=""):
    path = f"/home/jovyan/data/dataset/{version}/data{sufix}.csv"
    with open(path, 'w') as f:
      json.dump(entries, f)

def loadDataframe(version, sufix=""):
    dataset = getDataset(version, sufix)
    entries = dataToCSV(dataset, version, sufix)
    path = f"/home/jovyan/data/dataset/{version}/data{sufix}.csv"
    df = pd.read_csv(path, names=["text", "intent"], sep=";")
    return df

def filterDataframe(df):
    filteredColumns = [
        'REGISTRAR_TOMA_MEDICAMENTO',
        'REGISTRAR_ESTADO_EMOCIONAL',
        'REGISTRAR_CITA_MEDICA',
        'REGISTRAR_SALUDO',
        'REGISTRAR_DESPEDIDA'
    ]
    filter = df['intent'].isin(filteredColumns)
    dfFiltered = df[filter]
    return dfFiltered

def saveTrainTestDataset(version):
    dataset = getDatasetAsDictionaryText(version)
    df = loadDataframe(version)
    train=df.sample(frac=0.8,random_state=2) #random state is a seed value
    test=df.drop(train.index)
    saveDF(version, train, "train", dataset)
```

2

```
    saveDF(version, test, "test", dataset)
    dfTrain = loadDataframe(DATASET_VERSION, "_train")
    dfTest = loadDataframe(DATASET_VERSION, "_test")
    return dfTrain, dfTest

def saveDF(version, df, split, dataset):
    output = {}
    for index, row in df.iterrows():
        output[row["text"]] = dataset[row["text"]]

    toJSON = []

    for item in output.keys():
        toJSON.append(output[item])


    path = f"/home/jovyan/data/dataset/{DATASET_VERSION}/data_{split}.json"
    with open(path, "w", encoding="utf-8") as f:
        json.dump(toJSON, f, ensure_ascii=False, indent=4)
        print(f"Saved at {path}")
```

[2]:
```
DATASETS = {}
DATATRAIN = {}
DATATEST = {}
```

1.1 Conjunto v1

[3]:
```
DATASET_VERSION = "v1" # Versión del conjunto de datos
%run -i dataset_generate.py prod "v1"
```

```
-----------
DATASET GENERATE
-----------
Dataset table: tfm-prod-dialogs
Version from: None
Version to: v1
Dataset None: 0 entries
Getting dialogs from tfm-prod-dialogs, please wait…
Found 598 dialogs.
Dataset v1: 499 entries
Copy to /home/jovyan/data/dataset/v1/data.json
Error: /home/jovyan/data/dataset/v1 already exists
```

[4]:
```
df = loadDataframe(DATASET_VERSION)
df.head()
```

[4]:
```
                                                text  \
0  Espero que no se enamore de ti como en Her, au…
1  Hola estoy muy sola necesito de tus consejos y…
2  Este mismo mes de junio a las 8'30 con el enfe…
3  Hola, he estado un poco ocupado, lo siento, in…
4                       el rinialer me toca a las 9h

                        intent
0             NO_IDENTIFICADO
1  REGISTRAR_ESTADO_EMOCIONAL
2       REGISTRAR_CITA_MEDICA
3  REGISTRAR_ESTADO_EMOCIONAL
4  REGISTRAR_TOMA_MEDICAMENTO
```

[5]:
```
df.shape
```

[5]:
```
(176, 2)
```

[6]:
```
df["intent"].unique()
```

[6]:
```
array(['NO_IDENTIFICADO', 'REGISTRAR_ESTADO_EMOCIONAL',
       'REGISTRAR_CITA_MEDICA', 'REGISTRAR_TOMA_MEDICAMENTO',
       'REGISTRAR_NECESIDAD', 'REGISTRAR_DESPEDIDA', 'REGISTRAR_SALUDO',
       'REGISTRAR_ALABANZA', 'REGISTRAR_CONFIRMACION',
       'CONSULTAR_ESTADO_PERSONA_MAYOR', 'REGISTRAR_COVID',
       'REGISTRAR_SITUACION_ADVERSA', 'ANOTAR_TOMA_MEDICAMENTO',
       'ACTUALIZAR_TOMA_MEDICAMENTO', 'REGISTRAR_MEDIDA_MEDICA',
       'REGISTRAR_SINTOMA', 'REGISTRAR_ANECDOTA', 'REGISTRAR_DUDA',
       'REGISTRAR_DOMOTICA', 'RECORDAR_MEDICACION',
       'RECORDAR_CITA_MEDICA', 'REGISTRAR_ACTIVIDAD', 'CONSULTAR_TIEMPO',
       'REGISTRAR_TELEGRAM', 'REGISTRAR_RECHAZO', 'CONSULTAR_NUTRICION'],
      dtype=object)
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 176 entries, 0 to 175
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    176 non-null    object
 1   intent  176 non-null    object
dtypes: object(2)
memory usage: 2.9+ KB
```

[8]:
```
df.describe()
```

[8]:
```
         text           intent
count     176              176
unique    172               26
top      Hola  NO_IDENTIFICADO
freq        3               35
```

[9]:
```
df[df['text']=='Hola']
```

[9]:
```
      text            intent
16    Hola  REGISTRAR_SALUDO
136   Hola  REGISTRAR_SALUDO
161   Hola  REGISTRAR_SALUDO
```

[10]:
```
df['intent'].value_counts()
```

[10]:
```
NO_IDENTIFICADO                  35
REGISTRAR_TOMA_MEDICAMENTO       28
REGISTRAR_ESTADO_EMOCIONAL       26
REGISTRAR_CITA_MEDICA            17
REGISTRAR_SALUDO                 11
REGISTRAR_DESPEDIDA               9
REGISTRAR_ALABANZA                7
REGISTRAR_SITUACION_ADVERSA       6
REGISTRAR_DOMOTICA                5
REGISTRAR_NECESIDAD               4
REGISTRAR_CONFIRMACION            4
REGISTRAR_ANECDOTA                3
REGISTRAR_COVID                   3
CONSULTAR_ESTADO_PERSONA_MAYOR    3
REGISTRAR_MEDIDA_MEDICA           2
REGISTRAR_SINTOMA                 2
REGISTRAR_ACTIVIDAD               2
REGISTRAR_RECHAZO                 1
REGISTRAR_TELEGRAM                1
CONSULTAR_TIEMPO                  1
ACTUALIZAR_TOMA_MEDICAMENTO       1
RECORDAR_CITA_MEDICA              1
RECORDAR_MEDICACION               1
REGISTRAR_DUDA                    1
ANOTAR_TOMA_MEDICAMENTO           1
CONSULTAR_NUTRICION               1
Name: intent, dtype: int64
```

[11]:
```
dfFiltered = filterDataframe(df)
dfFiltered.head()
```

[11]:
```
                                                text  \
1  Hola estoy muy sola necesito de tus consejos y…
2  Este mismo mes de junio a las 8'30 con el enfe…
3  Hola, he estado un poco ocupado, lo siento, in…
4                      el rinialer me toca a las 9h
6               Ahora voy a cenar luego hablamos

                        intent
1  REGISTRAR_ESTADO_EMOCIONAL
2       REGISTRAR_CITA_MEDICA
3  REGISTRAR_ESTADO_EMOCIONAL
4  REGISTRAR_TOMA_MEDICAMENTO
6         REGISTRAR_DESPEDIDA
```

[12]:
```
dfFiltered.shape
```

[12]:
```
(91, 2)
```

[13]:
```
dfFiltered['intent'].value_counts()
```

[13]:
```
REGISTRAR_TOMA_MEDICAMENTO    28
REGISTRAR_ESTADO_EMOCIONAL    26
REGISTRAR_CITA_MEDICA         17
REGISTRAR_SALUDO              11
REGISTRAR_DESPEDIDA            9
Name: intent, dtype: int64
```

[14]:
```
dfTrain, dfTest = saveTrainTestDataset(DATASET_VERSION)
DATASETS["v1"] = df
DATATRAIN["v1"] = dfTrain
DATATEST["v1"] = dfTest
```

```
Saved at /home/jovyan/data/dataset/v1/data_train.json
Saved at /home/jovyan/data/dataset/v1/data_test.json
```

En el conjunto de datos hay una notable diferencia entre el número de registros de REGISTRAR_TOMA_MEDICAMENTO (28) y REGISTRAR_DESPEDIDA (9), por lo que se va a descartar para generar otro conjunto más balanceado.

1.2 Conjunto v2

Tras la generación de nuevos diálogos por parte de los usuarios y la mejora de la herramienta que genera el conjunto de datos a partir de los diálogos para que elimine registros duplicados, se procede a analizar el nuevo conjunto de datos generado.

[15]:
```
DATASET_VERSION = "v2" # Versión del conjunto de datos
%run -i dataset_generate.py prod "v1" "v2"
```

```
-----------
DATASET GENERATE
-----------
Dataset table: tfm-prod-dialogs
Version from: v1
Version to: v2
Dataset v1: 176 entries
Getting dialogs from tfm-prod-dialogs, please wait…
Found 598 dialogs.
Dataset v2: 499 entries
Copy to /home/jovyan/data/dataset/v2/data.json
Error: /home/jovyan/data/dataset/v2 already exists
```

[16]:
```
df = loadDataframe(DATASET_VERSION)
df.head()
```

[16]:

	text	intent
0	Espero que no se enamore de ti como en Her, au…	NO_IDENTIFICADO
1	Hola estoy muy sola necesito de tus consejos y…	REGISTRAR_ESTADO_EMOCIONAL
2	Saludos, robot!	REGISTRAR_SALUDO
3	Este mismo mes de junio a las 8'30 con el enfe…	REGISTRAR_CITA_MEDICA
4	Hola hoy Melo he pasado muy bien	REGISTRAR_ESTADO_EMOCIONAL

[17]:
```
df.shape
```

[17]:
```
(236, 2)
```

[18]:
```
len(df["intent"].unique())
```

[18]:
```
24
```

[19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 236 entries, 0 to 235
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    236 non-null    object
 1   intent  236 non-null    object
```

```
dtypes: object(2)
memory usage: 3.8+ KB
```

[20]:
```
df.describe()
```

[20]:
```
         text           intent
count     236              236
unique    226               24
top     Hola  NO_IDENTIFICADO
freq        4               41
```

[21]:
```
df[df['text']=='Hola']
```

[21]:
```
     text            intent
27   Hola  REGISTRAR_SALUDO
72   Hola  REGISTRAR_SALUDO
185  Hola  REGISTRAR_SALUDO
217  Hola  REGISTRAR_SALUDO
```

[22]:
```
df['intent'].value_counts()
```

[22]:
```
NO_IDENTIFICADO                  41
REGISTRAR_ESTADO_EMOCIONAL       37
REGISTRAR_DESPEDIDA              30
REGISTRAR_TOMA_MEDICAMENTO       30
REGISTRAR_CITA_MEDICA            29
REGISTRAR_SALUDO                 16
REGISTRAR_SITUACION_ADVERSA      10
REGISTRAR_ALABANZA                7
REGISTRAR_DOMOTICA                5
REGISTRAR_NECESIDAD               4
REGISTRAR_CONFIRMACION            4
REGISTRAR_SINTOMA                 3
REGISTRAR_ANECDOTA                3
REGISTRAR_COVID                   3
CONSULTAR_ESTADO_PERSONA_MAYOR    3
REGISTRAR_MEDIDA_MEDICA           2
REGISTRAR_ACTIVIDAD               2
CONSULTAR_TIEMPO                  1
REGISTRAR_RECHAZO                 1
REGISTRAR_TELEGRAM                1
ANOTAR_TOMA_MEDICAMENTO           1
RECORDAR_MEDICACION               1
REGISTRAR_DUDA                    1
CONSULTAR_NUTRICION               1
Name: intent, dtype: int64
```

[23]:
```
dfFiltered = filterDataframe(df)
dfFiltered.head()
```

[23]:
```
                                              text  \
1  Hola estoy muy sola necesito de tus consejos y…
2                                  Saludos, robot!
3  Este mismo mes de junio a las 8'30 con el enfe…
4                  Hola hoy Melo he pasado muy bien
5                                            Adiós

                        intent
1  REGISTRAR_ESTADO_EMOCIONAL
2           REGISTRAR_SALUDO
3      REGISTRAR_CITA_MEDICA
4  REGISTRAR_ESTADO_EMOCIONAL
5         REGISTRAR_DESPEDIDA
```

[24]:
```
dfFiltered.shape
```

[24]:
```
(142, 2)
```

[25]:
```
dfFiltered['intent'].value_counts()
```

[25]:
```
REGISTRAR_ESTADO_EMOCIONAL    37
REGISTRAR_DESPEDIDA           30
REGISTRAR_TOMA_MEDICAMENTO    30
REGISTRAR_CITA_MEDICA         29
REGISTRAR_SALUDO              16
Name: intent, dtype: int64
```

[26]:
```
dfTrain, dfTest = saveTrainTestDataset(DATASET_VERSION)
DATASETS["v2"] = df
DATATRAIN["v2"] = dfTrain
DATATEST["v2"] = dfTest
```

```
Saved at /home/jovyan/data/dataset/v2/data_train.json
Saved at /home/jovyan/data/dataset/v2/data_test.json
```

El conjunto de datos está balanceado y listo para entrenar, por lo tanto, si se algún modelo entrenado con este conjunto alcanza el 85% de tasa de acierto, este conjunto de datos **sería válido**.

2 Entrenamiento de modelos

[27]:
```
import os
from google.cloud import dialogflow

PROJECT_ID = "cuidadormayores"
```

```
SESSION = "123456789"
LANGUAGE_CODE = "es"

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "/home/jovyan/credentials/google/
↪key.json"

def identifyIntentWithDialogflow(text):
    session_client = dialogflow.SessionsClient()
    session = session_client.session_path(PROJECT_ID, SESSION)
    text_input = dialogflow.TextInput(text=text, language_code=LANGUAGE_CODE)
    query_input = dialogflow.QueryInput(text=text_input)
    response = session_client.detect_intent(
        request={"session": session, "query_input": query_input}
    )
    intent = response.query_result.intent.display_name
    return intent
```

[28]:
```
def getValidator(languajeModel):
    if languajeModel == "dialogflow":
        return identifyIntentWithDialogflow

def validate(test, languajeModel):
    validator = getValidator(languajeModel)
    total = 0
    success = 0
    intents = ['REGISTRAR_TOMA_MEDICAMENTO', 'REGISTRAR_ESTADO_EMOCIONAL',␣
↪'REGISTRAR_CITA_MEDICA', 'REGISTRAR_SALUDO', 'REGISTRAR_DESPEDIDA']
    for item in test:
        expectedIntent = item["intent"]
        if expectedIntent in intents:
            # En Google Action, el intent NO_IDENTIFICADO es el intent por␣
↪defecto, que
            # conservamos con el nombre que Google le da, por legibilidad.
            if "NO_IDENTIFICADO" == expectedIntent:
                expectedIntent = "Default Fallback Intent"
            text = item["text"]
            receivedIntent = validator(text)
            if expectedIntent == receivedIntent:
                success += 1
            else:
                print(f"FAILED: expected: {expectedIntent}, received:␣
↪{receivedIntent} on text: {text}")
            total += 1
    accuracy = round(success/total*100, 2)
    return total, success, accuracy
```

2.1 Modelo Dialogflow: Dataset v2

[29]:
```
DATASET_VERSION = "v2"
LANGUAJE_MODEL = "dialogflow"
# Versión del conjunto de datos a utilizar: v2
%run -i dialogflow.py "v2"
```

```
-----------
DIALOGFLOW GENERATE
-----------
Dataset version: v2
Language model: dialogflow
Error: /home/jovyan/data/dialogflow/v2 already exists
```

A continuación:

- Eliminar todos los intents del proyecto de Dialogflow, excepto el "DefaultFallbackIntent".
- Importar en el proyecto de Dialogflow el dataset generado, comprimido en .zip (no añadir model_info.json).
- Eliminar el intent "NO_IDENTIFICADO", puesto que contiene textos que, por ahora, no han sido clasificados, de modo que deberían ir a "DefaultFallbackIntent".

[30]:
```
test = getDataset(DATASET_VERSION, "_test")
total, success, accuracy = validate(test, LANGUAJE_MODEL)

print(f"Total: {total}, success: {success}, accuracy: {accuracy}%")
```

```
FAILED: expected: REGISTRAR_DESPEDIDA, received: Default Fallback Intent on
text: Pues ya no te cuento nada más
FAILED: expected: REGISTRAR_DESPEDIDA, received: Default Fallback Intent on
text: Creo que es todo para hoy
FAILED: expected: REGISTRAR_SALUDO, received: REGISTRAR_CITA_MEDICA on text:
Hola hoy me han quitado los puntos
Total: 29, success: 26, accuracy: 89.66%
```

El modelo **es válido**, puesto que el criterio de evaluación exige una tasa de acierto superior al 85%, y este modelo ha alcanzado un **89.66%**.

A.3. Encuesta inicial a los usuarios

Encuesta sobre un BOT para cuidar y acompañar a personas mayores

¡Buenas!

¿Te apetece participar en un experimento para ayudarme con mi TFM? Necesito pares persona mayor / persona joven donde la persona mayor tiene que comunicarse escribiéndose con un BOT en Telegram y contarle qué medicinas se toma, cómo se encuentra y cualquier otra necesidad.

Por ahora, necesito que me rellenéis esta encuesta juntos con datos reales y tengáis instalado Telegram. Más adelante os enviaré el enlace al BOT, al que iréis viendo evolucionar. No te asustes de sus respuestas, porque, a veces, es muy imaginativo.

Sólo si os apetece, sin compromiso y hasta que queráis.

Por supuesto, en la conversación no necesito datos reales, me interesa la interacción usuario y BOT. ¡En el formulario sí que necesito que los datos sean reales!

¡Muchas gracias!

*Obligatorio

1. Correo *

2. ¿Cuál es el nombre de la persona mayor?

3. ¿Y el de la persona joven?

4. Para la persona mayor, ¿eres hombre o mujer? *

   Marca solo un óvalo.

   - ◯ Hombre
   - ◯ Mujer
   - ◯ Prefiero no decirlo

5. ¿Qué edad tiene la persona mayor? *

   _____

6. ¿Qué edad tienes la persona joven? *

   _____

7. ¿Qué aplicaciones de mensajería utilizáis? *

   Selecciona todos los que correspondan.

   - ☐ WhatsApp
   - ☐ Telegram
   - ☐ Line
   - ☐ Otro: _____

8. Para la persona mayor, ¿crees que un BOT en tu cliente de mensajería te puede ayudar y acompañar? *

   Marca solo un óvalo.

   - ◯ Sí
   - ◯ No

9. Para la persona joven, ¿crees que un BOT puede ayudar y acompañar a la persona mayor encuestada? *

Marca solo un óvalo.

- Sí
- No

10. ¿Te gustaría que el BOT tuviese iniciativa para escribirte sin que le preguntes nada? *

Marca solo un óvalo.

- Sí
- No

11. ¿Te ayudaría que el BOT conociese tu estado de ánimo? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada						Mucho

12. ¿Te ayudaría que el BOT te recordase medicamentos? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada						Mucho

13. ¿Te ayudaría que el BOT te recordase citas médicas? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

14. ¿Te ayudaría que el BOT te preguntase de vez en cuando cómo te sientes? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

15. ¿Te ayudaría que el BOT te avisase si la persona mayor de la que cuidas no responde? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

16. ¿Te ayudaría si el BOT te informase si la persona mayor se siente triste o sola? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

17. ¿Te gustaría poder contar al BOT tus anécdotas, que las recordara y pudiera hablarte de ellas? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

18. ¿A qué más pensáis que os podría ayudar el BOT? También puedes dejar cualquier otro comentario sobre las preguntas anteriores.

A.4. Encuesta final a los usuarios

Encuesta sobre un BOT para cuidar y acompañar a personas mayores

¡Buenas!

¡Hemos llegado a la última fase de mi TFM! Muchas gracias por haber participado. Es posible que el BOT te haya escrito estos días para preguntarte cómo estás, o te haya informado de cómo está la persona a tu cuidado o recordado una cita médica o medicamento a tomar. Si no es así, ¡podéis probarlo ahora! Esto es lo que el BOT entiende:
- Registrar la toma de un medicamento: Tengo que tomarme este medicamento en este momento.
- Registrar una cita médica: Tengo que ir este día a esta cita con este especialista.
- Registrar el estado en el que te encuentras: Estoy triste, me siento alegre...

Cuando indiquéis un medicamento o una cita al BOT, éste os avisará los 15 minutos anteriores a la toma del medicamento de la cita.

Cuando registréis vuestro estado, la persona que cuida podrá preguntar cómo está la persona a su cuidado. En caso de que una persona mayor registre un estado de tristeza o soledad, la persona al cuidado será informada.

Si la persona mayor lleva varias horas sin hablar con el BOT, le preguntará cómo está. Si no responde, acabará avisando a la persona que le cuida.

Por favor, como última fase de mi TFM, os pido que juguéis un poco con el BOT y rellenéis esta última encuesta de satisfacción. Cuando tenga vuestra encuesta desactivaré al BOT para que no os hable más y concluir así las pruebas con los usuarios finales.

¡Muchísimas gracias por haber participado!

*Obligatorio

1. Correo *

2. ¿Cuál es el nombre de la persona mayor? *

3. ¿Y el de la persona joven? *

4. Para la persona mayor, después de probar el BOT, ¿crees que te puede ayudar y acompañar? *

   Marca solo un óvalo.

   - Sí
   - No

5. Para la persona joven, después de probar el BOT, ¿crees que ha sido útil para ayudar y acompañar a la persona mayor? *

   Marca solo un óvalo.

   - Sí
   - No

6. ¿Te ha parecido bien el BOT te haya escrito por iniciativa propia? *

   Marca solo un óvalo.

   - Sí
   - No

7. ¿Te ha ayudado que conozca tu estado de ánimo? *

   Marca solo un óvalo.

	1	2	3	4	5	
Nada						Mucho

8. ¿Te ha ayudado a recordar medicamentos? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

9. ¿Te ha ayudado a recordar citas médicas? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

10. ¿Te ha ayudado que te pregunte de vez en cuando cómo te sientes? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

11. ¿Te ha ayudado que te avise si la persona mayor de la que cuidas lleva horas sin responder? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

12. ¿Te ha ayudado que el BOT te informase cuando la persona mayor se ha sentido triste o sola? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

13. En una escala del 1 al 5, ¿podéis dar una valoración global de tu satisfacción con el BOT? *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

14. ¿Tienes algún otro comentario que hacer?