UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

# AGENTE VIRTUAL PARA AYUDA AL SITIO WEB DE EOL

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

JOSHUA EZEQUIEL NÚÑEZ ESCUDERO

PROFESOR GUÍA:
JUAN MANUEL BARRIOS NÚÑEZ

MIEMBROS DE LA COMISIÓN:
FELIPE BRAVO MARQUEZ
VALENTÍN MUÑOZ APABLAZA

SANTIAGO DE CHILE
2023

# Resumen

La Oficina de Educación Online (EOL) ofrece servicios a una gran comunidad de estudiantes, desarrollando nuevos cursos y metodologías para mejorar la educación. EOL cuenta con su propio sitio web que ellos mismos administran, pero actualmente se enfrenta a un problema recurrente: el equipo de la mesa de ayuda es reducido y recibe una gran cantidad de solicitudes, lo que resulta en una carga excesiva para ellos. Por esta razón, han buscado la mejor alternativa para solucionar este problema: crear un sistema que pueda solucionar automáticamente los problemas de los estudiantes y usuarios que se comunican con la mesa de ayuda.

Desde un enfoque práctico y teórico, se analizan y etiquetan los datos ingresados a la mesa de ayuda durante un año. Se crea una API REST que funciona como *backend* para la administración de la solución, y se diseña, configura y analiza un modelo de inteligencia artificial que se alimenta de estos datos. Este modelo permite responder de forma eficiente a las inquietudes más recurrentes de la plataforma, y permitirá avanzar hacia una solución eficaz.

La evaluación de este sistema se centra principalmente en el estudio del modelo de inteligencia artificial. Se busca la mejor configuración y métricas utilizando los datos recopilados y etiquetados. Los modelos constan de variaciones entre las etiquetas utilizadas, y se exploran distintas técnicas de procesamiento de lenguaje natural, tales como *word embeddings* y *stopwords*.

EOL evalúa el sistema, conformado por un equipo de distintos integrantes de diversas áreas, quienes se muestran satisfechos con los resultados obtenidos. No solo valoran las métricas de desempeño, sino también la posibilidad de entrenar el modelo con nuevos datos y tener total administración sobre el sistema. El modelo alcanza una exactitud del 70 %, y utiliza técnicas como *stemming* y *TF-IDF*. Para comprobar que el modelo tiene un buen rendimiento, se aplica validación cruzada, obteniendo un resultado de exactitud promedio del 66.8 %.

*A mi familia y amigos que siempre han confiado en mis capacidades aún más que yo.*

# Agradecimientos

Cuando ingresé a la universidad nunca me imaginé lo largo que sería este camino, fue difícil y doloroso. Si no hubiese sido por mi familia que me estuvo apoyando incondicionalmente y mis amigos que han sido pilares en mis momentos álgidos no sé de qué forma lo hubiera conseguido. Recordando el día de inducción siento y sé que ha pasado mucho, agradezco todo lo que me ha tocado vivir pese a las dificultades, me han hecho la persona que soy hoy, agradezco mucho la experiencia y conocimientos adquiridos.

A mi Papá que siempre tiene alguna palabra sabia con la que animarme, a mi Mamá que siempre me da ánimos para que no me rinda, ambos han sido esenciales, gracias por su amor incondicional en estos 25 años. siempre han estado pidiéndole a Dios por mi a quien también agradezco porque siempre me he sentido protegido y en lo pequeño he visto la oraciones de mis cercanos, me ha dado calma cuando lo he necesitado.

A todos mis amigos que son muchos, quienes me han soportado y demostrado su cariño año tras año. Agradecer especialmente a mi amigo Jota, no sé si se imagina lo mucho que me ayudó en los últimos 3 años, pero ha sido un verdadero pilar, me ha ayudado como nadie, ha estado en momentos duros y también en los alegres, tiene un corazón muy grande y es muy preocupado, espero que le vaya bien en todo lo que se propone y le estaré agradecido toda la vida. A mi amigo Winsi, que siempre busca la forma de subir el ánimo y ayudar para que uno pueda para ver el vaso medio lleno, siempre dispuesto a escuchar y aportar. A mi amiga Alondra que aunque suele estar subiendo cerros, siempre está cuando necesito desahogarme, me anima, me escucha y entiende. Los 3 han sido súper especiales en este último tiempo y sin su ayuda no podría estar escribiendo finalmente estos últimos párrafos de mi memoria.

Quiero agradecer también a Gabo por su incondicionalidad, amistad y preocupación. Lalo por tantos años seguidos de amistad que aún perduran. Karen porque me soportó y es una buena amiga. Bryan que sufrió junto a mi desde la primera R.

Son tantos que quiero dejar al menos el nombre de la mayoría gracias a mis amigos, Cesar, Gonzalo, Braulio, Nano, Cata, Diego, Winni, Joaco, Izi, Celso, Ivan, a toda la familia Carillo y a los que se me puedan olvidar pero siempre han estado.

Gracias a todos por tanto por el solo hecho de estar ahí siempre serán los mejores.

# Tabla de Contenido

# Índice de Tablas

# Índice de Figuras

# Capítulo 1

# Introducción

La importancia de la comunicación en todo tipo de relaciones es innegable. Específicamente, en el contexto de una empresa u organización, una comunicación adecuada con su público objetivo puede generar numerosas ventajas, como la prevención y solución de conflictos, el fortalecimiento de la imagen y reputación corporativa, y la fidelización de los clientes.

En este contexto, las tecnologías están avanzando rápidamente con el objetivo de mejorar y hacer más expedita la comunicación. Debido a las crecientes necesidades de ofrecer servicios de atención al cliente y resolución de todo tipo de problemas y duda, desde el siglo pasado se han comenzado a desarrollar Agentes Virtuales, también conocidos como *Chatbots*. Un *Chatbot* se define como un programa computacional similar a un correo electrónico que puede responder a los mensajes de usuarios de computadores [1].La finalidad de estos agentes es delegar el trabajo humano a un asistente virtual, agilizar las soluciones de las problemáticas más comunes de los usuarios y aliviar las funciones del área de soporte de las organizaciones.

Estos Agentes Virtuales han sido objeto de investigación, perfeccionamiento e implementación durante las última decádas. En la actualidad, existen soluciones empresariales en la nube, como *Azure*, que ofrecen sus servicios con modelos preentrenados en Procesamiento de Lenguaje Natural (NLP, por sus siglas en inglés) para que, a través de una capa adicional de entrenamiento, puedan operar como Agentes Virtuales. Sin embargo, el costo de estos servicios puede ser elevado y no todas las empresas tienen acceso a ellos. Por lo tanto, implementar un servicio de soporte virtual customizado puede ser la opción más viable para aquellas empresas que buscan mejorar su comunicación con el público a través de medios digitales.

Para desarrollar un Agente Virtual efectivo se necesitan habilidades y conocimientos en diversas áreas, incluyendo NLP, Redes Neuronales y *Machine Learning*. No obstante, además de los conocimientos técnicos, es necesario tener una comprensión profunda del contexto empresarial para poder crear una solución adaptada a las necesidades de la empresa en cuestión. En consecuencia, la construcción de un Agente Virtual implica una combinación de habilidades técnicas y conocimientos específicos del negocio que permitan crear una herramienta de comunicación eficaz y personalizada.

La Oficina de Educación Online (EOL) tiene como principal objetivo desarrollar capa-

cidades al interior de la Universidad de Chile para enriquecer la práctica docente mediante la implementación de nuevas metodologías de educación mediadas por tecnología. En este sentido, EOL se ha convertido en la plataforma de gestión del aprendizaje de la Universidad, diseñada específicamente para ser utilizada en cursos impartidos en formato completamente en línea o semi presencial.

EOL es responsable de la administración de aproximadamente 10 plataformas, la mayoría de las cuales están basadas en *openedx*, una plataforma de educación en línea de código abierto desarrollada por la Universidad de Harvard y el MIT. Actualmente, todas estas plataformas son atendidas por una única persona en el servicio de mesa de ayuda. Si bien no se han presentado problemas de capacidad, EOL tiene previsto lanzar una plataforma masiva en inglés para estudios internacionales en un futuro próximo. Como resultado, se espera que el volumen de solicitudes de soporte aumente significativamente, por lo que contar con un Agente Virtual se convierte en una necesidad inminente.

En la misma línea, EOL ha notado que el tipo de preguntas realizadas por los estudiantes se repite con frecuencia y que, para la mayoría de ellas, se pueden proporcionar respuestas genéricas (se estima que cerca del 90 % de los *tickets* que llegan a mesa de ayuda se solucionan de ese modo). Como resultado, el trabajo del integrante de la mesa de ayuda a menudo implica copiar y pegar respuestas, lo que representa una carga innecesaria para su trabajo diario. Con la implementación de un Agente Virtual, se espera poder automatizar este proceso, lo que permitirá a los estudiantes obtener respuestas rápidas y precisas a sus preguntas, y liberará al personal de la mesa de ayuda para atender casos más complejos y críticos.

La finalidad del presente trabajo es desarrollar un sistema o Agente Virtual que pueda permitirle a EOL responder automáticamente a consultas comunes de los usuarios. Este sistema permitirá escalar automáticamente los requerimientos más complejos al integrante de la mesa de ayuda, según el tema y la naturaleza de las solicitudes, lo que ayudará a optimizar la gestión de solicitudes y mejorar la eficiencia del servicio de atención al cliente.

## 1.1. Objetivo General

El objetivo de esta memoria es desarrollar un software de Inteligencia Artificial (IA) que facilite la resolución de *tickets* de los usuarios que acceden a la mesa de ayuda. Para esto primero se analizarán las solicitudes ingresadas por usuarios durante el último año junto con las respuestas dadas por el personal de soporte, para luego desarrollar un agente que se pueda adaptar a estas y a nuevas solicitudes.

## 1.2. Objetivos Específicos

1. Recopilar solicitudes o tickets con las preguntas recibidas por EOL para la clasificación y entrenamiento de un modelo.
2. Explorar las solicitudes más comunes, estudio de tópicos y definición de preguntas más comunes.
3. Creación de un etiquetador de tickets ya que estos no se encuentran clasificados.
4. Desarrollar un agente virtual específico para EOL y que permita reentrenamiento con nuevos datos. Que entregue una respuesta automatizada a la solicitud de soporte de los usuarios.
5. Evaluar el modelo para mejorarlo en base a los datos obtenidos y procesados durante el periodo.

# Capítulo 2

# Marco de trabajo

## 2.1. Utilidad de un agente

Desde la década de 1960, se ha estado trabajando en el desarrollo de Agentes Virtuales o *chatbots*, cuyo objetivo principal es simular la comunicación humana [7] de tal manera que las personas no puedan distinguir si están interactuando con una máquina o con un ser humano. Con el tiempo, el uso de los agentes virtuales se ha expandido en diversas áreas, incluyendo la educación, la recuperación de información, los negocios y el comercio electrónico, entre otros.

Aunque el objetivo original de los Agentes Virtuales es imitar la comunicación humana, en la actualidad también se busca que cumpla su función de manera precisa respecto a un área específica sin la importancia de imitar tan bien la comunicación humana tanto como la precisión con la que responde correctamente los temas del área. En el caso particular de los *chatbots* de empresas, es de vital importancia que sean capaces de resolver de manera óptima los *tickets* de asistencia y reducir el esfuerzo humano.

Los resultados de las evaluaciones de distintos Agentes Virtuales con distintos objetivos han mostrado que a los usuarios les resultó fácil de usar y satisfactorio [17]. Los usuarios aprecian la capacidad de expresar sus necesidades en su propio idioma y sentir que la computadora hace el trabajo por ellos. Además, consideraron que el agente redujo el tiempo de interacción en comparación con la respuesta de un ser humano.

El objetivo de un desarrollador de *chatbots* debería ser construir herramientas que ayuden a las personas a interactuar con las computadoras de manera simple y eficiente mediante el uso de recursos de lenguaje natural, pero no reemplazar completamente el papel humano o imitar perfectamente la conversación humana.

## 2.2. Tipos de agentes

Los agentes pueden clasificarse en diferentes tipos según el problema a resolver: (1) el dominio del conocimiento, (2) el servicio que proporcionan, (3) los objetivos, (4) el procesamiento de entrada y el método de generación de respuestas, (5) la necesidad o no de ayuda humana y (6) el método de construcción [4].

1. La clasificación basada en el dominio del conocimiento se refiere al conocimiento al que puede acceder el agente virtual o la cantidad de datos con los que se entrena. Existen dos tipos de agentes según este criterio: los de dominio abierto y cerrado. Los agentes de dominio abierto están entrenados para hablar sobre temas generales, mientras que los agentes de dominio cerrado están preparados para hablar sobre temas más específicos.

2. La clasificación basada en el servicio prestado se enfoca en la tarea que el agente virtual realiza y en la cercanía emocional que tiene con el usuario. Se pueden distinguir entre agentes interpersonales e intrapersonales. Los primeros están diseñados para entregar servicios y su función principal es proporcionar información de manera concisa y eficiente. Por otro lado, los *bots* intrapersonales están centrados en la interacción con el usuario, lo que genera una conversación más personal entre el humano y el *bot*, con el objetivo de hacer que la persona se sienta comprendida y acompañada. Un ejemplo claro de esto son los asistentes virtuales, que mantienen una conversación continua ya que almacenan la mayor cantidad de información posible sobre el usuario.

3. La clasificación basada en los objetivos se refiere al propósito principal que los agentes virtuales buscan lograr. Existen varios objetivos, como informar, conversar o realizar tareas.

4. La clasificación basada en el método de procesamiento de entradas y generación de respuestas considera el método utilizado para procesar las entradas y generar respuestas. Existen tres modelos que se utilizan para generar respuestas adecuadas: el modelo basado en reglas, el modelo basado en recuperación y el modelo generativo.

    - Los agentes de modelos basados en reglas son la arquitectura más común en la construcción de los primeros *chatbots*. Este tipo de *bot* es el más simple, ya que está limitado por las reglas de programación y es poco flexible en cuanto a las diferencias gramaticales y contextuales.
    - Por otro lado, el modelo basado en recuperación ofrece más flexibilidad al consultar y analizar recursos disponibles a través de APIs. Un agente virtual basado en recuperación analiza varios candidatos de respuesta de un índice antes de aplicar un filtro de coincidencia y elegir la mejor respuesta.
    - Finalmente, el modelo generativo es el más completo y similar a la comunicación humana. Este modelo se basa en mensajes antiguos y nuevos utilizando técnicas de aprendizaje automático y *deep learning*. A pesar de ser costosos y difíciles de construir y entrenar, son los más efectivos en cuanto a la precisión de la respuesta.

5. La clasificación basada según la cantidad de ayuda humana necesaria en sus componentes. Por un lado, los *bots* asistidos requieren que un ser humano intervenga en alguna

etapa para que puedan seguir funcionando, mientras que los *bots* independientes son completamente automáticos en todo su flujo sin requerir la intervención humana.

6. Los Agentes Virtuales también pueden clasificarse según los permisos proporcionados por su plataforma de desarrollo. Estas pueden ser de código abierto o cerrado. Aunque el código abierto brinda mayor flexibilidad a los desarrolladores, en el caso de una plataforma con un flujo de datos muy grande, un *bot* de código cerrado se verá beneficiado de estos permisos.

Para esta memoria, es importante seleccionar un Agente Virtual que se adapte adecuadamente al problema que se busca resolver. El Agente Virtual elegido debería ser capaz de especializarse en diferentes áreas de EOL. Por lo tanto, se utilizará la clasificación basada en el dominio, específicamente un agente de dominio cerrado. En términos del servicio prestado, se necesitará un agente interpersonal que pueda solucionar problemas y, por lo tanto, sus objetivos estarán centrados en informar, aunque también en algunos casos específicos deberá ser capaz de realizar tareas como generar un ticket en el sistema.

En cuanto al método de procesamiento de entradas, aunque el modelo generativo es el ideal, debido a los plazos de entrega de la memoria, un modelo basado en recuperación es una opción más realista y adecuada para el proyecto. Además, se requerirá una cantidad mínima de ayuda humana para extender y reentrenar los datos a medida que los usuarios generen nuevas problemáticas y frases.

Por último, en cuanto a los permisos, EOL trabaja con proyectos de código abierto, por lo que el agente virtual seleccionado también seguirá esta misma línea.

## 2.3. Resultados actuales

En proyectos similares se han logrado resultados exitosos. Por ejemplo, en un caso en el que los usuarios debían elegir entre enviar correos electrónicos o utilizar un chat en vivo, se encontró que los usuarios preferían el chat en vivo. Esto llevó a la empresa a asignar personal para responder en tiempo real, pero también generó problemas adicionales, ya que los administradores debían dedicar tiempo a responder y los clientes tenían que esperar por la respuesta, lo que resultaba inconveniente.

Existen diversos Agentes disponibles, pero algunos requieren que los usuarios configuren manualmente las frases clave. En el ejemplo, se desarrolló un *chatbot* de preguntas frecuentes (FAQ) que respondía automáticamente a los clientes mediante el uso de una red neuronal recurrente (RNN) con memoria a largo plazo (LSTM) para la clasificación de texto. Los resultados experimentales demostraron que el agente podía reconocer el 86,36 % de las preguntas y responder con *accuracy* del 93,2 % [12].

## 2.4. Conceptos

### 2.4.1. Procesamiento de texto

#### Stemming, Lematización y Stop Words

Stemming es una estrategia que consiste en reducir una palabra a su origen o palabra raíz, por tanto todas las variaciones de una palabra se ven reducidas a la misma palabra raíz, como ejemplo las palabras conducir, conduce, conduciendo, se reducen a un mismo stem conduc [14].

La Lematización también corresponde a una estrategia de reducción, sin embargo no a la raíz de la palabra si no que al lema de esta para desambiguar y estandarizar, un ejemplo sería para las palabras corre, corrió, corriendo, se reducen al lema correr.

Stopwords corresponden a palabras que no aportan con información al contexto de una frase, carecen de sentido si se escriben por si solas tales como conjunciones, artículos, preposiciones y adverbios, estas palabras se quitan del vocabulario de entrenamiento de un modelo para facilitar la comprensión de contextos [8].

#### Word embeddings

Word Embeddings[11] son un conjunto de modelos de lenguaje NLP que permiten representar numéricamente las palabras de un vocabulario, y por tanto representar frases con vectores numéricos, esto es específicamente útil para Machine learning, en esta memoria se usan dos tipo de Word Embeddings, estos son One Hot Encoding y TF-IDF.

#### One Hot Encoding

También conocido solo como One-Hot[11] es la estrategia de Word embedding más simple de realizar, este modelo de lenguaje define los vectores con valores de 1 o 0 dependiendo de que la palabra se encuentre en la frase o no, por tanto es el menos preciso ya que no afecta la relevancia de cada palabra, todas equivalen pero tiene la ventaja de ser muy sencillo de implementar.

#### TF-IDF

La teoría detrás de TF-IDF [16] se basa en la combinación de dos factores: la frecuencia de términos (TF) y la frecuencia inversa de documentos (IDF). El primero se utiliza para medir cuántas veces aparece un término en un documento, mientras que el segundo corresponde al factor inverso de la frecuencia con que un término aparece en los documentos, asignando un valor menor cuanto mayor sea esta frecuencia. De esta manera, TF-IDF se define como el producto entre ambos factores (ver Def. 2.3).

**Definición 2.1** *Term Frequency*

- *Sea* $freq_{ij}$ *la frecuencia del termino* $k_i$ *en el documento* $d_j$

$$tf_{ij} = 1 + log(freq_{ij})$$

**Definición 2.2** *Inverse Document Frequency*

- *N el número de documentos del dataset y* $n_i$ *el número de documentos donde aparece* $k_i$

$$idf_i = log(N/n_i)$$

**Definición 2.3** *TF-IDF*

- $w_{ij}$ *es el peso del término i para el documento j*

$$w_{ij} = tf_{ij} * idf_i$$

### 2.4.2. Machine learning

#### Red neuronal Feedforward

Una red neuronal puede ser de distintos tipos, las redes multicapa Feedforward[10] que se usan en esta memoria son el tipo de red neuronal más básico y corresponden a las redes que no son cíclicas, se dividen en varias capas teniendo una capa de entrada y una salida, el flujo siempre es desde la capa de entrada a la capa de salida.

#### Overfitting

El Overfitting[9] o sobreajuste es un problema que ocurre cuando se entrenan redes neuronales. Ocurre cuando se entrena tanto un modelo que este se ajusta demasiado a las características más específicas de los datos de entrenamiento y por tanto pierde el foco de las características relevantes, si se prueba el modelo con datos los del entrenamiento tendrá una alta tasa de éxito en su repuesta, sin embargo para nuevos ejemplos no tendrá las capacidades de predicción o clasificación, esto significa que el modelo más que aprender solo memorizó.

#### Métricas de entrenamiento

Para validar que un modelo está efectivamente bien entrenado es necesario examinar las métricas del entrenamiento [15] ya que con solo una métrica no es posible asegurar que el modelo no cometa fallas.

### Loss

Corresponde al error de la predicción e indica que tan alto es este, por tanto mientras menor sea mejor es para el modelo, significa que el error de aproximación es menor, en *machine learning* se busca minimizar esta métrica.

### Accuracy

De forma informal se puede definir Accuracy como la razón entre la cantidad de predicciones correctas y el total de predicciones.

**Definición 2.4** *Definición informal Accuracy*

$$Accuracy = \frac{Cantidad\ predicciones\ correctas}{Total\ predicciones}$$

Sin embargo la definición de Accuracy que se usa para cuantificar la exactitud de los modelos de machine learning se calcula en base a los términos positivos y negativos donde TP son los Verdaderos Positivos, VN los Verdaderos Negativos, FP los Falsos Positivos y FN los Falsos Negativos.

**Definición 2.5** *Definición Accuracy*

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

El problema de la métrica Accuracy es que cuando los conjuntos de datos no están equilibrados no es posible determinar que el valor obtenido sea un modelo efectivamente bien entrenado, puede tener una alta exactitud, sin embargo como se trata de conjuntos desbalanceados puede fallar siempre en los casos con menos datos.

### *Precision* y *Recall*

La *Precision* corresponde a la proporción de verdaderos positivos detectados correctamente entre todos los casos positivos falsos y verdaderos.

**Definición 2.6** *Definición Precision*

$$Precision = \frac{TP}{TP + FP}$$

*Recall* corresponde a la proporción de verdaderos positivos se detectaron correctamente.

**Definición 2.7** *Definición Recall*

$$Recall = \frac{TP}{TP + FN}$$

En términos de métricas no sirve obtener una sin la otra, por lo que para evaluar un modelo efectivamente es necesario obtener ambas, en general una mejora en la *Precision* suele reducir el valor del *Recall* y viceversa. Es por ello que existe una métrica que equivale a una razón entre ambas.

#### F1-Score

Corresponde a la razón entre las métricas de *Precision* y *Recall*, se considera media armónica de estos.

**Definición 2.8** *Definición F1-Score*

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

#### Validación cruzada

En la validación cruzada[6], se divide el conjunto de datos de entrenamiento en partes iguales para usarlas como conjunto de validación en cada iteración. De esta manera, se puede verificar que las métricas obtenidas no se deben a una selección específica de datos como conjunto de validación y prueba. Esto ayuda a asegurar que el modelo sea capaz de generalizar de manera adecuada a nuevos datos.

## 2.5. Tecnologías disponibles

En el ámbito de la creación de modelos en Python, una biblioteca muy popular en el campo del *machine learning* es Tensorflow [3]. Esta herramienta permite la creación de una variedad de modelos, con diferentes tipos de capas de neuronas, lo que la convierte en una solución muy versátil.

Además de Tensorflow, existe otra herramienta muy útil en el ámbito del data science llamada Scikit-Learn [13]. Esta herramienta permite la clasificación, regresión y clustering de datos. Además, Scikit-Learn es compatible con otras bibliotecas de Python, como NumPy, SciPy y matplotlib.

Una de las funciones que se pueden realizar con Scikit-Learn es el uso de TfidfVectorizer. Esta herramienta permite calcular el vocabulario de un conjunto de frases, mediante TF-IDF.

Scikit-Learn también ofrece otra herramienta útil llamada KFold, la cual es un método utilizado para implementar validación cruzada [6] en modelos de *machine learning* con el fin de evaluar su desempeño. La validación cruzada es una metodología ampliamente utilizada debido a su capacidad para realizar remuestreo en modelos predictivos con el fin de estimar el verdadero error de predicción.

Keras, una funcionalidad proporcionada por Tensorflow, permite la creación de redes neuronales, lo cual es imprescindible en el mundo actual. Una red neuronal es un modelo matemático o aritmético que simula una red neuronal real y está compuesta por neuronas artificiales que reciben un conjunto de parámetros de entrada, se activan mediante una función y procesan la entrada para devolver una respuesta. La estructura, dimensiones y configuraciones de la red neuronal dependen de la salida esperada y los datos de entrada. En cuanto al problema de clasificación para múltiples clases, también existe una solución con redes neuronales [5].

Una red neuronal tiene varios beneficios, como la capacidad de obtener resultados muy buenos si se configura correctamente y se entrena con buenos datos. Además, las pruebas y la validación son rápidas, y es un área de estudio innovadora. Sin embargo, el entrenamiento puede ser lento y costoso, lo que puede ser una desventaja si no se disponen de los recursos necesarios.

# Capítulo 3

# Solución propuesta

## 3.1. Sistema actual

El sistema de ayuda actual se basa en la plataforma OTRS, que es un sistema de gestión de *tickets*. Los correos electrónicos que llegan a una cuenta de Gmail se dirigen a través de un túnel directamente al sistema de administración de *tickets*. Esta plataforma genera informes semestrales que cuantifican la cantidad de *tickets* recibidos, su resolución y el tiempo que se tardó en resolverlos.

Gran parte de las consultas que llegan al sistema de soporte son relacionadas con problemas de acceso, en particular con mensajes como "se me bloqueó el acceso por demasiados intentos" y "mi clave no funciona y estoy seguro de que es esa". EOL estima que cerca del 90 % de los *tickets* que llegan son de este tipo y no requieren necesariamente de interacción humana para resolverlos, por lo que la implementación de un Agente Virtual sería beneficioso. Cabe destacar que la página web de EOL tampoco cuenta con un espacio de preguntas frequentes (FAQ).

## 3.2. Etapa inicial

Para este trabajo se necesitan datos que consisten en frases documentadas del tipo {pregunta : respuesta}, y que cubran diversos temas y sus variaciones, siendo diferenciados por especialidad. Por ejemplo, para el área de estudios internacionales, se incluyeron preguntas y respuestas específicas para esa sección. Actualmente, EOL solo requiere que el Agente Virtual admita el idioma español, aunque otros idiomas podrían ser necesarios en el futuro.

En primer lugar, se llevó a cabo un proceso de análisis de datos, se analizó el funcionamiento del formulario de contacto y los *tickets* que se han almacenado desde que se implementó el sistema. Este análisis es imprescindible para clasificar, limpiar, y determinar correctamente la frecuencia de los datos que se utilizarán como entrada antes de aplicar una solución con

IA.

Para la limpieza de datos se desarrolló un *script* de Python que utiliza expresiones regulares para formatear los textos. De esta manera, se eliminan las palabras y frases que no aportan valor y se obtiene un texto reducido que rescata el contenido fundamental del mensaje. Esto permite generar un modelo más preciso a partir de los nuevos datos.

Etiquetar los datos es un trabajo laborioso que requiere de esfuerzo humano. Para abordar esta tarea se desarrolló una interfaz sencilla en Python, utilizando la biblioteca *pyQT5*, que facilita el etiquetado y puede ser compartida con aquellos que tengan acceso para participar en el proceso. Los datos de entrada y de salida consisten en archivos de valores separados por comas (csv).

## 3.3. Etapa final

La etapa final comprende la definición del Agente Virtual, la elección de los algoritmos y la evaluación de cuál se adapta mejor al tamaño y tipo de datos. A partir de las métricas de desempeño, se seleccionará el modelo que tenga un mejor rendimiento. El Agente Virtual se encargará de solucionar las dudas que no requieren interacción humana, de modo que el personal encargado de responder pueda enfocarse en aquellos *tickets* que realmente necesitan atención humana.

El objetivo es poner el sistema en producción al final de este trabajo, comenzando a interactuar con usuarios finales y recopilando información relevante sobre su funcionamiento y desempeño. La estrategia de desarrollo consistirá en crear una Interfaz de Programación de Aplicaciones (API) de *backend* que contenga la lógica del modelo entrenado y se comunique con el *frontend* a través de llamadas de invocación. En este proceso, EOL exige la realización de pruebas unitarias como buena práctica.

El proyecto utilizará Python como lenguaje de programación, Django como *framework* para el *backend* y React para el *frontend*. La idea es que el Agente Virtual se integre en una ventana de tipo *widget* que se pueda incorporar eficientemente a la web (aunque esto no es un requisito y el Agente puede tener su propia ruta para su interfaz). Para lograr una comunicación en tiempo real, se utilizarán *websockets* y una base de datos persistente como Redis. Para almacenar las frases, usuarios, entre otros, se necesitará una base de datos SQL. Se espera que el servicio del Agente se comunique con los datos internos del sistema para identificar a los usuarios y personalizar el trato del bot.

La página web actual donde se pretende integrar el chat es accesible a través de una ruta pública (`https://eduonline.ing.uchile.cl/`), donde los estudiantes deben autenticarse con sus credenciales. La incorporación de un Agente Virtual en la plataforma proporcionaría un espacio de preguntas frecuentes y de contacto. Sin embargo, debido a las limitaciones técnicas de tamaño en los servidores de EOL, se decidió alojar los servicios de manera independiente en Heroku [2] es una plataforma como servicio que admite varios lenguajes de programación. Una de las primeras plataformas en la nube.

# Capítulo 4

# Tratamiento de los datos

El formato de los datos entregados por EOL se observa en la Tabla 4.1. En total, fueron recolectados cerca de 14.000 mensajes de este tipo, a los cuales se les realizaron distintas tareas relacionadas con la obtención, limpieza y etiquetado de los datos.

Tabla 4.1: Estructura de los datos de EOL.

| **ticket_id** | **ticket_number** | **title** | **create_time** | **change_time** | **cola** | **estado_ticket** | **name** | a_body |
|---|---|---|---|---|---|---|---|---|
| id del ticket | número del ticket | titulo del correo | fecha de creacion | fecha de creacion | origen del ticket | estado de la respuesta del ticket (cerrado, abierto) | nombre que la plataforma le asignó al ticket | contenido del ticket. corresponde al cuerpo de un correo electrónico |

- ticket_id: Los mensajes están organizados por su ticket_id, este no es único y se repite en cada interacción. Por ejemplo, si para un problema en específico se enviaron y recibieron cinco correos, entonces todos ellos comparten el mismo ticket_id y el mismo ticket_number.
- ticket_number: Corresponde a un identificador interno de EOL, se comporta igual que el ticket_id pero estos no tienen el mismo valor.
- title: Corresponde al título que la plataforma de contacto o el formulario asigna a la eventualidad, este luego es enviado como asunto del correo para su contestación.
- create_time: Corresponde a la fecha en que el mensaje fue enviado.
- change_time: Corresponde a la fecha en que el mensaje fue modificado.
- cola: Corresponde al origen del *ticket*, este puede ser EOL, Ayuda, Docencia en Línea, Help-open o Zoom Uchile.
- estado_ticket: Corresponde al estado actual del proceso, puede ser un *ticket* cerrado satisfactoriamente o no.
- name: Es otro título designado por la plataforma para diferenciar estos *tickets*.
- a_body: Es la columna donde se encuentra el contenido del mensaje, el formato de este texto es plano y contiene todo el contenido del correo, incluyendo el mensaje específico y el contenido de cualquier mensaje del hilo acumulativamente.

Teniendo en cuenta la estructura de los datos, se presentan múltiples dificultades para llevar a cabo un análisis efectivo de los mensajes, las cuales se detallan a continuación:

1. No están identificados los mensajes en el orden del hilo.
2. Las columnas nombre y cola no son suficientemente concisas para usarlas como etiquetas válidas para un modelo.
3. Los datos de los mensajes se encuentran guardados en la columna a_body, incluyen todos los tag HTML de los correos en los que ocurrió la interacción usuario-EOL.
4. El cuerpo del mensaje incluye mucha información poco relevante para un análisis efectivo (saludos y despedidas del correo).
5. El cuerpo del mensaje incluye todo el contenido del hilo de los correos.

## 4.1. Pasos realizados

Para solucionar los problemas anteriores se desarrolló un *script* de Python donde se toma el csv y se preprocesan sus datos para luego generar un nuevo csv con nuevas columnas. Las tareas realizadas por el script son las siguientes:

1. Se pasaron todos los mensajes a minúsculas para evitar problemas con comparaciones de texto.
2. Se eliminaron los caracteres sobrantes de cada mensaje (doble espacios, saltos de línea, caracteres especiales).
3. Se eliminó cada saludo de cada correo.
4. Se eliminó cada despedida de cada correo y también el contenido del hilo.
5. Se reemplazaron los correos de cada mensaje por el *flag* [EMAIL].
6. Se reemplazaron los rut de cada mensaje por el *flag* [RUT].
7. Se reemplazaron los numero de *ticket* de cada mensaje por el flag [TICKET].
8. Se reemplazaron los números telefónicos de cada mensaje por el flag [PHONE].
9. Se agregó una columna con el nuevo mensaje limpio.
10. Se eliminaron los mensajes sin contenido luego de la limpieza de los textos.
11. Se exportó el nuevo csv para su uso posterior.

Para realizar el etiquetado de los datos se desarrolló una interfaz gráfica mediante la biblioteca *pyQT5* de Python, que permite a los usuarios etiquetar manualmente los mensajes (Figura 4.1). Esta herramienta de etiquetado manual facilita la tarea de etiquetado de los datos y es de gran utilidad en la siguiente fase del proceso.

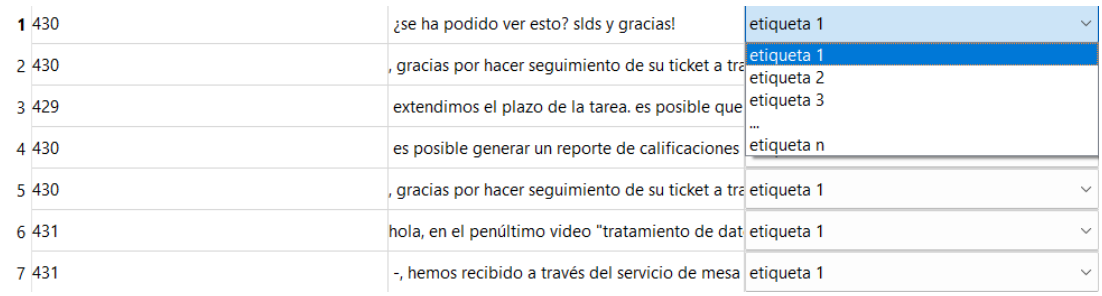


Figura 4.1: Primera versión del etiquetador.

En la primera versión del etiquetador, el proceso resultaba poco práctico ya que se tenía que seleccionar manualmente la misma etiqueta para cada mensaje dentro del ticket, lo cual resultaba tedioso. Además, en la pantalla se mostraban varios tickets a la vez, lo que dificultaba la selección y etiquetado de cada mensaje. Estos problemas fueron solucionados en una segunda versión del etiquetador (Figura 4.2).

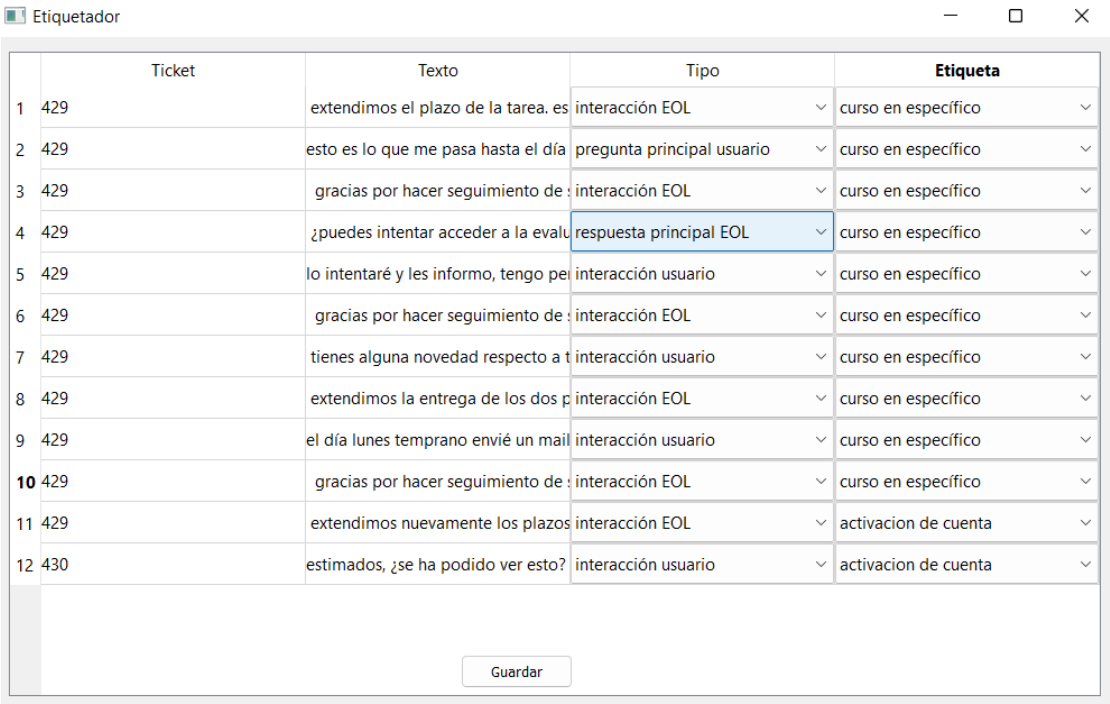


Figura 4.2: Versión final del etiquetador.

Se tuvo que etiquetar los datos dos veces debido a un error de código que causó la pérdida de las etiquetas previas al cerrar y volver a abrir el archivo. A pesar de esto, el proceso de etiquetado fue exitoso. Además, dado que se trataba de una gran cantidad de *tickets* y mensajes, se contó con la colaboración de externos en el proceso de etiquetado, quienes firmaron un acuerdo de confidencialidad generado por EOL para garantizar la responsabilidad en el tratamiento de los datos etiquetados.

# Capítulo 5

# Base de Datos

## 5.1. Motor de base de datos

En un principio se optó por utilizar una base de datos *SQLite3* debido a la facilidad de implementación. Esta base de datos es embebida, lo que significa que toda la información se encuentra en un único archivo, y no requiere instalaciones locales. A pesar de contar con la mayoría de las funcionalidades de una base de datos relacional *SQL*, su desempeño se ve limitado en el manejo de grandes cantidades de datos y su capacidad de tipos de datos es reducida. Para escalar la aplicación a un ambiente productivo con consultas más complejas y costosas, se hace necesario migrar la base de datos a un motor más avanzado.

La base de datos final utilizada corresponde a *PostgreSQL* alojada en Heroku para que sea accesible en todo momento y no consuma los recursos locales del equipo donde se desarrolla la aplicación. La elección de *PostgreSQL* se debió a que tiene mejor desempeño y ofrece más funcionalidades que *SQLite3*, lo que permitirá realizar consultas más complejas y costosas en un futuro.

## 5.2. Modelo de datos

La intención inicial de la base de datos era almacenar los datos etiquetados que estaban en cuatro archivos CSV diferentes proporcionados a los encargados del etiquetado. Para ello, se propuso un modelo centrado en la entidad **Frases**, donde cada frase representa un mensaje de interacción entre el usuario y la mesa de ayuda. Cada *ticket* tiene varias frases, y cada frase tiene una etiqueta y un tipo de mensaje. Aunque los tipos de mensaje son fijos, se crearon las entidades **Etiquetas** y **TipoMensaje** para permitir su administración y preparar el modelo para futuros cambios. Además, se añadió una columna llamada "Eliminado" a las tres entidades con el propósito de prevenir la eliminación de datos de manera física. Esto se logró mediante la inclusión de un valor booleano que indica si el registro ha sido eliminado o no. Este método, conocido como *"soft delete"*, ayuda a evitar la pérdida de información.

Para permitir la visualización de la tabla de Frases en esta memoria se seleccionaron algunos registros con texto acotado.

Tabla 5.1: Datos de la tabla Frases.

| Id | TicketId | Cuerpo | FechaCrea | TipoMensajeId | EtiquetaId | Eliminado |
|---|---|---|---|---|---|---|
| 78 | 443 | solucionado | 2020-01-13 12:34:22.000 | 1 | 14 | false |
| 125 | 453 | procesando | 2020-01-17 14:34:08.000 | 2 | 7 | false |
| 208 | 476 | prueba | 2020-03-05 14:30:18.000 | 1 | 14 | false |
| 229 | 482 | nan | 2020-03-12 03:21:19.000 | 1 | 1 | false |
| 394 | 528 | 1 | 2020-03-16 19:18:39.000 | 1 | 10 | false |
| 398 | 529 | a | 2020-03-16 19:33:01.000 | 1 | 7 | false |
| 405 | 530 | sii muchas | 2020-03-16 20:54:04.000 | 1 | 7 | false |
| 474 | 549 | alumno inscrito. | 2020-03-17 19:25:13.000 | 2 | 4 | false |
| 547 | 570 | nan | 2020-03-18 12:57:10.000 | 1 | 7 | false |
| 554 | 572 | nan | 2020-03-18 12:24:08.000 | 1 | 6 | false |

La tabla de **Etiquetas** tiene registros eliminados, estos son los que se ignoran para el entrenamiento de un nuevo modelo.

Tabla 5.2: Datos de la tabla Etiquetas.

| Id | Etiqueta | Eliminado |
|---|---|---|
| 1 | activacion de cuenta | true |
| 2 | ayuda con certificados | false |
| 3 | cambio de correo | true |
| 4 | creacion de curso | false |
| 5 | creacion de usuario | true |
| 6 | contraseña | true |
| 7 | cuenta uchile | false |
| 8 | cuenta zoom | false |
| 9 | cuenta bloqueada | true |
| 10 | curso en específico | false |
| 11 | inicio de sesión | true |
| 12 | inasistencia | true |
| 13 | pagos | true |
| 14 | otros | false |

Los datos de la tabla **TipoMensaje** existentes con los que se etiquetaron los datos actualmente son los que se pueden ver en la tabla 5.3.

Tabla 5.3: Datos de la tabla TipoMensaje.

| Id | Tipo | Eliminado |
|---|---|---|
| 1 | interacción usuario | false |
| 2 | interacción EOL | false |
| 3 | pregunta principal usuario | false |
| 4 | respuesta principal EOL | false |

Debido a que la entidad **Frases** contiene dos llaves foráneas, se creó la vista **VFrases** que contiene los datos de Etiquetas y TipoMensaje con su representación en caracteres y no su identificador. Esta vista sirve para mostrar el mantenedor de frases de modo que el usuario final pueda visualizar los datos por sus nombres y no por sus identificadores numéricos, lo que mejora la comprensión de la información. Una primera versión de la base de datos se observa en la Figura 5.1.

Tabla 5.4: Datos de la vista VFrases.

| Id | TicketId | Cuerpo | Tipo | Etiqueta | Eliminado |
|---|---|---|---|---|---|
| 3.083 | 1.307 | ¿y en u-cursos? | interacción usuario | otros | false |
| 5.089 | 1.961 | ya, muchas | interacción usuario | otros | false |
| 10.732 | 3.631 | ya contestado | interacción usuario | curso en específico | false |
| 9.836 | 3.409 | todo ok. | interacción usuario | otros | false |
| 11.027 | 3.724 | todo ok | interacción EOL | curso en específico | false |
| 4.286 | 1.693 | ticket repetido | interacción usuario | otros | false |
| 852 | 661 | ticket listo | interacción usuario | otros | false |
| 2.057 | 1.006 | ticket cerrado | interacción EOL | curso en específico | false |
| 988 | 701 | ticket cerrado | interacción EOL | cuenta zoom | false |
| 858 | 663 | ticket cerrado | interacción usuario | otros | false |

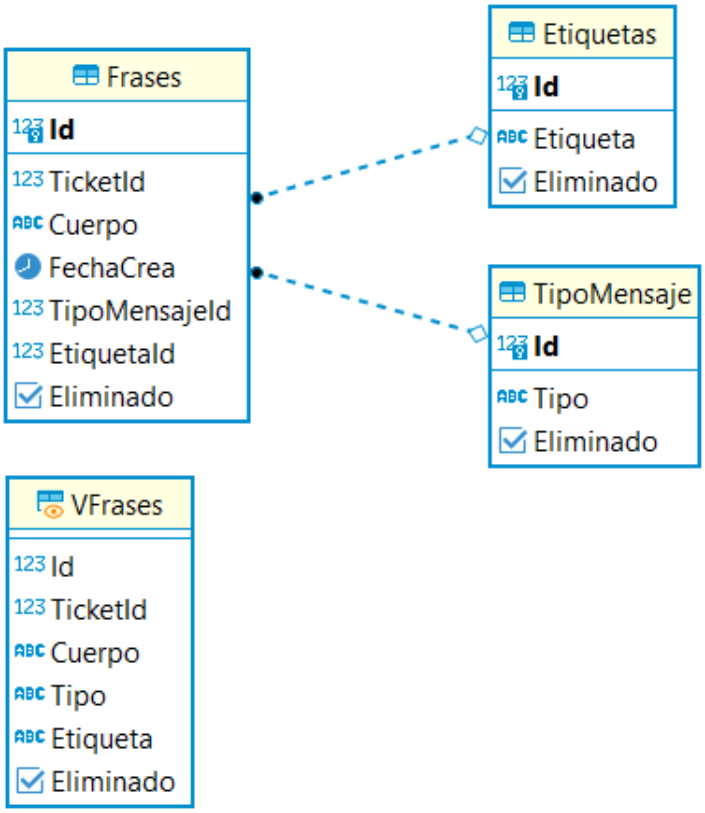


Figura 5.1: Primera versión de la base de datos.

Para que el *backend* estuviera conectado a la base de datos, fue necesario añadir la en-

tidad de **Usuarios** para implementar la seguridad de *tokens* y autenticación para solicitar las credenciales de un usuario en cada petición. La columna Usuario almacena el nombre de usuario y la columna Clave almacena la contraseña encriptada desde el backend. Las columnas Nombre, ApellidoPaterno, ApellidoMaterno y Email se utilizan para guardar los datos personales del usuario. LastLogin guarda la fecha de la última conexión del usuario, mientras que Bloqueado es un *flag* que indica si el usuario está bloqueado o no. La segunda versión de la base de datos se observa en la Figura 5.2.

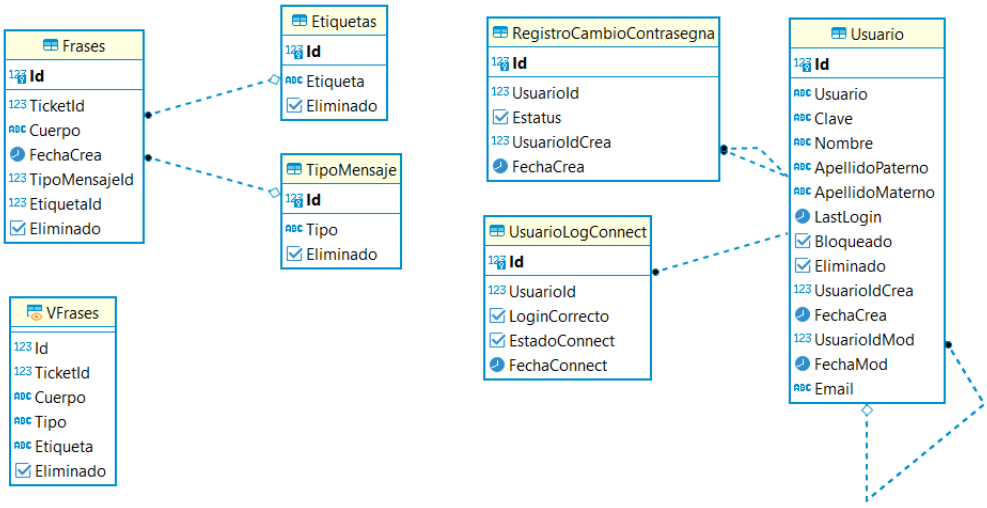


Figura 5.2: Segunda versión de la base de datos

Las tablas **RegistroCambioContrasegna** y **UsuarioLogConnect** si bien no se usarán en esta etapa del proyecto, se crearon para mejoras futuras donde se pueda permitir el cambio de contraseñas y bloquear cuentas luego de una cierta cantidad de login fallidos.

Tabla 5.5: Datos de la tabla Usuario.

| Id | Usuario | Clave | Nombre | ApellidoPaterno | ApellidoMaterno | LastLogin | Bloqueado |
|---|---|---|---|---|---|---|---|
| 1 | admin | pbkdf2_sha256$320000$zXxAx... | Administrador | | | 2023-03-11 05:31:21.824 | false |

La última modificación relevante a la base de datos fue la inclusión de la entidad **Modelos** (Figura 5.3), que tiene el objetivo de guardar los nombres con los que se guardan lo modelos, vocabularios y clases con que se entrenó cierto modelo, la fecha en que fue creado, un *flag* para marcar un modelo como activo y todas las métricas obtenidas durante el entrenamiento.

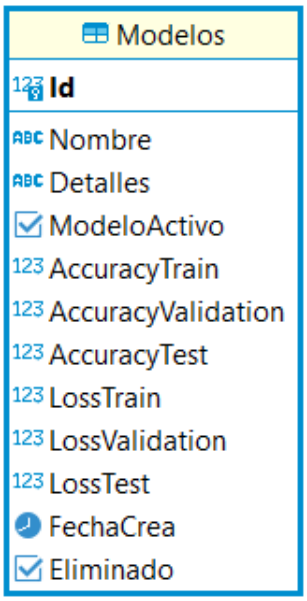


Figura 5.3: Tabla para los modelos entrenados.

Las columnas **AccuracyTrain**, **AccuracyValidation**, **AccuracyTest**, **LossTrain**, etc. Almacenan los resultados promedio de cada métrica obtenida en el entrenamiento del modelo, el nombre del modelo que se guarda en la columna **Nombre** corresponde al nombre con el que el modelo, vocabulario y clases se almacenan localmente como versiones.

Tabla 5.6: Datos de la tabla Usuario.

| Id | Nombre | Detalles | ModeloActivo | AccuracyTrain | AccuracyValidation | AccuracyTest | LossTrain |
|---|---|---|---|---|---|---|---|
| 30 | model_2023-03-11-18-53-20-928907.h5 | | false | 0,9952456355 | 0,6497175097 | 0,6653234334 | 0,0256308354 |
| 29 | model_2023-03-11-18-49-10-280056.h5 | | false | 0,9857369065 | 0,7062146664 | 0,6945271414 | 0,0389213376 |
| 28 | model_2023-03-11-18-47-03-255453.h5 | | false | 0,9873217344 | 0,5932203531 | 0,5723122323 | 0,0498885885 |
| 27 | model_2023-03-11-18-45-12-864933.h5 | | false | 0,9889065027 | 0,6723163724 | 0,6023459012 | 0,0402098224 |
| 26 | model_2023-03-11-18-43-18-292872.h5 | | false | 0,990491271 | 0,6666666865 | 0,6623324552 | 0,0325897597 |
| 25 | model_2023-03-11-18-40-53-068665.h5 | | false | 0,9920760989 | 0,6610169411 | 0,6483271223 | 0,0330781154 |
| 24 | model_2023-03-11-17-18-06-658046.h5 | | false | 0,9857369065 | 0,6610169411 | 0,6484281212 | 0,0452359319 |
| 23 | model_2023-03-11-17-09-06-770937.h5 | | false | 0,9952456355 | 0,7118644118 | 0,7381129239 | 0,0250077806 |
| 22 | model_2023-03-11-17-06-17-364035.h5 | | false | 0,9920760989 | 0,7175140977 | 0,7103948572 | 0,0257947892 |
| 21 | model_2023-03-11-16-48-09-790789.h5 | | false | 0,9920760989 | 0,6440678239 | 0,6029238123 | 0,0223394651 |
| 20 | model_2023-03-11-12-20-15-314718.h5 | | false | 0,9950544238 | 0,6384180784 | 0,5981223412 | 0,0201092791 |
| 19 | model_2023-03-11-12-17-44-513435.h5 | | false | 0,9762046337 | 0,4976303279 | 0,4765123542 | 0,0835897252 |
| 18 | model_2023-03-11-04-18-35-332899.h5 | | false | 0,9797739387 | 0,5213270187 | 0,5082172372 | 0,072630249 |
| 17 | model_2023-03-11-04-17-32-725225.h5 | | false | 0,9821534753 | 0,5071089864 | 0,4872331234 | 0,0649921522 |
| 16 | model_2023-03-11-04-13-14-925081.h5 | | false | 0,9910979271 | 0,7005649805 | 0,6881231244 | 0,024428321 |
| 15 | model_2023-03-11-04-05-44-982020.h5 | | false | 0,9841521382 | 0,6553672552 | 0,6712381232 | 0,041268114 |

Finalmente, el modelo incluye todas las entidades mencionadas anteriormente, como se puede ver en la Figura 5.4

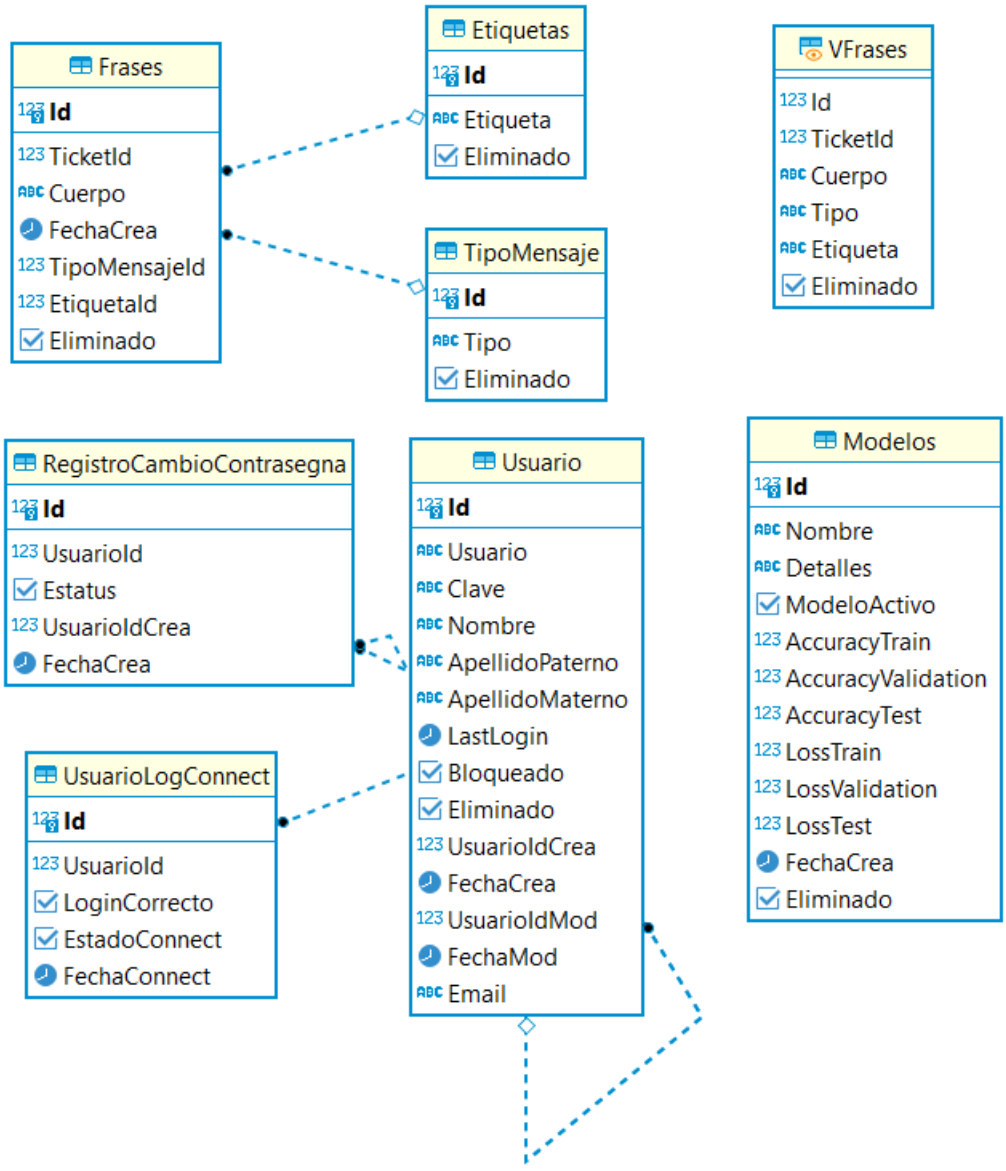


Figura 5.4: Modelo final de la base de datos.

# Capítulo 6

# Desarrollo de la API

Se optó por desarrollar una solución basada en un *backend* que funcionara como una API, específicamente una API REST (Transferencia de Estado Representacional), debido a su capacidad de adaptarse a cualquier tipo de *frontend* o *backend*, sin limitaciones de tecnologías o lenguajes, utilizando únicamente el protocolo HTTP.

Las respuestas se encuentran actualmente almacenadas en un diccionario de formato {id_etiqueta: respuesta}, son *dummy* para esta versión del sistema pero es importante agregarlas al modelo de datos o simplemente mejorar el etiquetado para usar las respuestas ya almacenadas.

```
respuestas = {
    1: 'respuesta para activacion de cuenta',
    2: 'respuesta para ayudar con su certificado',
    3: 'respuesta para ayudar con cambio de correo',
    4: 'respuesta para creacion de curso',
    5: 'respuesta para creacion de usuario',
    6: 'respuesta para ayuda con la contrasegna',
    7: 'respuesta para ayuda con la cuenta uchile',
    8: 'respuesta para ayuda con la cuenta zoom',
    9: 'respuesta para ayuda con cuenta bloqueada',
    10: 'respuesta para curso en especifico',
    11: 'respuesta para ayuda con inicio de sesion',
    12: 'respuesta para inasistencias',
    13: 'respuesta para ayuda con los pagos',
    14: 'respuesta para cualquier otro tipo de mensaje',
}
```

Listing 6.1: Diccionario de respuestas

A continuación se realiza un resumen de la funcionalidad y funciones que tiene cada apartado en la API sin profundizar en el formato de entrada o salida.

## 6.1. Versiones de metodologías

El objetivo principal de este trabajo es evaluar el rendimiento de un Agente Virtual en función de la inteligencia que utiliza y cómo el usuario final interpreta los resultados. Por lo tanto, se decidió probar tres tipos de lógica, cada una con un nivel de inteligencia diferente:

1. Expresiones regulares, que son la forma más básica de inteligencia artificial. Esta lógica consiste en una serie de reglas y patrones que identifican las frases para determinar qué respuesta corresponde dar.
2. Utilización de los datos recolectados para generar un vocabulario por cada tipo de frase y luego, a través de medidas numéricas, encontrar la clase que mejor se aproxima.
3. Desarrollo de un modelo de *machine learning* que utiliza las frases y clases para el entrenamiento del modelo.

- POST **/api/[vn]/messages/**: La obtención de respuestas está dado por las versiones a las que se quieran consultar donde vn corresponde a la versión (v1, v2, v3).
- POST **/api/[vn]/train/**: El entrenamiento de las versiones 2 y 3 se ejecutan haciendo un llamado a esta ruta, donde vn corresponde a la versión (v2, v3). Para v2 actualiza los vocabularios por etiqueta. Para v3 crea un nuevo modelo, lo guarda localmente y guarda tanto su identificador como sus métricas en las base de datos.

Se decidió crear una aplicación de **Django** para implementar las tres versiones de métodos. La primera versión solo requiere una ruta para enviar el mensaje del usuario y devolver la respuesta correspondiente capturada por la expresión regular que mejor se ajuste. Para las últimas versiones, se requería la implementación de un mantenedor de frases que permitiera extender el sistema, lo cual se explica en la próxima sección.

## 6.2. Mantenedores

### 6.2.1. Mantenedor de frases

Tener la posibilidad de agregar, modificar o eliminar frases es indispensable para un sistema que utiliza estos datos como base de sus predicciones.

- GET **/api/mantainer/dataphrases/**: Permite acceder a la visualización de las Frases gracias a la vista VFrases.
- GET **/api/mantainer/frases/[Id]**: Obtiene los datos de las Frases, si se extiende la url pasándole el Id permite obtener la frase correspondiente a ese Id.
- POST **/api/mantainer/frases/**: Permite la creación de una nueva Frase.
- UPDATE **/api/mantainer/frases/[Id]**: Actualiza los datos de una Frase.
- DELETE **/api/mantainer/frases/[Id]**: Elimina lógicamente una Frase.

### 6.2.2. Mantenedor de etiquetas

Para garantizar la sostenibilidad a largo plazo del sistema, es esencial permitir la administración de las etiquetas existentes, lo que considera la capacidad de agregar, quitar o modificarlas. Aunque las etiquetas actuales se seleccionaron en base a los mensajes disponibles y a la solicitud de EOL, es importante tener en cuenta que los requisitos pueden cambiar con el tiempo y, por lo tanto, el sistema debe ser capaz de adaptarse a estos cambios.

- GET **/api/mantainer/tag/[Id]**: Obtiene los datos de las Etiquetas, si se extiende la url pasándole el Id permite obtener la Etiqueta correspondiente a ese Id.
- POST **/api/mantainer/tag/**: Permite la creación de una nueva Etiqueta.
- UPDATE **/api/mantainer/tag/[Id]**: Actualiza los datos de una Etiqueta.
- DELETE **/api/mantainer/tag/[Id]**: Elimina lógicamente una Etiqueta.

### 6.2.3. Mantenedor de tipos de mensaje

Hasta ahora el etiquetado de los datos permite la elección de cuatro tipos de mensajes: (1) mensaje de usuario, (2) mensaje de EOL, (3) mensaje principal del usuario, y (4) mensaje principal de EOL.

Estos tipos de mensajes se crearon inicialmente para etiquetar y localizar el mensaje principal de la cadena de mensajes que conforman un *ticket*. Sin embargo, para asegurar la flexibilidad del sistema en el futuro, se contempla la posibilidad de agregar, editar o eliminar estos tipos de mensajes.

- GET **/api/mantainer/type/[Id]**: Obtiene los datos de los tipos de mensaje, si se extiende la url pasándole el Id permite obtener el tipo de mensaje correspondiente a ese Id.
- POST **/api/mantainer/type/**: Permite la creación de un nuevo tipo de mensaje.
- UPDATE **/api/mantainer/type/[Id]**: Actualiza los datos de un tipo de mensaje.
- DELETE **/api/mantainer/type/[Id]**: Elimina lógicamente un tipo de mensaje.

### 6.2.4. Mantenedor de Modelos

La idea de un mantenedor de modelos es permitir la administración de los modelos que se han entrenado en el sistema, visualizar los resultados que se han obtenido, elegir un modelo como activo o simplemente eliminar modelos.

- GET **/api/mantainer/model/[Id]**: Obtiene los datos de los modelos, si se extiende la url pasándole el Id permite obtener el modelo correspondiente a ese Id.

- UPDATE **/api/mantainer/model/[Id]**: Actualiza un modelo permitiendo marcarlo como Modelo activo.
- DELETE **/api/mantainer/model/[Id]**: Elimina lógicamente un modelo.

## 6.3. Seguridad

Con el fin de garantizar la seguridad de la API y evitar la exposición de información privada de los usuarios en la red, se ha implementado un sistema de inicio de sesión mediante JWT. Esto permite que el token sea requerido en cada solicitud a la API y tenga un tiempo de vigencia limitado. Una vez que el token ha expirado, se debe solicitar uno nuevo para continuar con el acceso a la API.

- POST **api/auth/login/**: Permite la obtención de un token JWT.

Con la obtención de este token en todas las demás peticiones anteriormente mencionadas se les debe incluir este token como validación **Bearer**.

# Capítulo 7

# Métodos utilizados

Para aumentar la versatilidad del *backend* se fueron dejando como versiones las progresiones de los tres métodos utilizados (expresiones regulares, TF-IDF, y *multiclass classification*) para la captura del texto recibido por usuarios, clasificando estos *inputs*.

## 7.1. Expresiones regulares

La primera versión consiste en una serie de reglas basadas en expresiones regulares, estas reglas sencillas se encargan de buscar secuencias de texto que cumplan con cierto formato y permitan diferenciar entre una etiqueta u otra.

## 7.2. TF-IDF

La segunda versión desarrollada consiste en crear un vocabulario separado por cada clase utilizando el método TfidfVectorizer. Este método crea un vector con los pesos de cada palabra para un conjunto de frases y se almacena localmente un vocabulario por cada clase. Al consultar por una nueva frase, se cargan los vocabularios de cada etiqueta y se calculan los descriptores de la nueva frase con cada vocabulario. Luego, se utiliza la similitud coseno para obtener la similitud más alta entre todos los vocabularios. Aunque este enfoque tiene casos de éxito, también tiene muchas fallas y, por lo tanto, se procede a probar otro método.

## 7.3. Multiclass classification

La última versión estudiada se trata de un clasificador, una red neuronal entrenada con los datos de cada categoría. Se probaron distintas configuraciones para la obtención de las mejores métricas. Para este tipo de problema, donde se tienen múltiples clases, la elección del mejor modelo se determinó según: (1) Precisión del conjunto de validación, (2) Precisión del

conjunto de entrenamiento, (3) Loss del conjunto de validación, y (4) Precisión del conjunto de testeo.

Debido a que el objetivo de este trabajo es desarrollar un clasificador de consultas para entregar las respuestas correctas se optó por un modelo del tipo **multiclass classifier**. Este modelo se caracteriza por permitir el entrenamiento con múltiples clases y no se limita a un modelo de positividad o negatividad, sino que entrega un porcentaje de precisión para un input respecto a cada clase con la que fue entrenado. Esto se logra creando una red neuronal **feedforward** consta de un modelo *Sequential*, que cuenta con una capa inicial con una cantidad de neuronas igual a las dimensiones del vocabulario; recibe un input con las *embeddings* de cada frase; tres capas intermedias que usan la función de activación **relu** de 256, 128 y 64 neuronas respectivamente; y una capa de salida con una cantidad de neuronas igual a la cantidad de etiquetas utilizadas en el entrenamiento, utiliza la función de activación **softmax** para obtener resultados entre 0 y 1, que representan una ponderación porcentual de cada clase.

La última capa incluye el optimizador conocido como **Stochastic Gradient Descent** o también **SDG** con un *learning rate* de 0.05 para llegar al máximo de entrenamiento tras 200 *epochs*. Este optimizador fue elegido tras probar su desempeño en comparación con el optimizador **adam** y notar una mejoría de 3 % en promedio en el conjunto de validación y test.

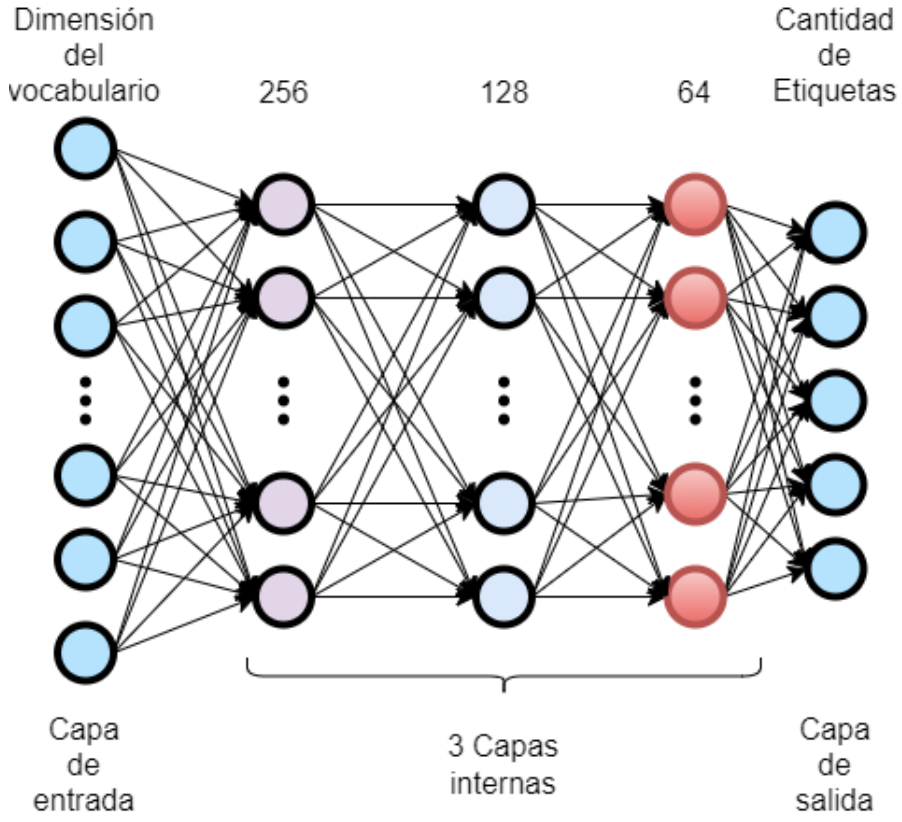


Figura 7.1: Arquitectura de la red neuronal usada.

# Capítulo 8

# Interfaces del sistema

Aunque la interfaz de usuario no es esencial para EOL, se creó un *frontend* como demostración de las principales características del sistema (Figura 8.1). La primera pantalla es un formulario de inicio de sesión que consume la ruta de autenticación de la API y almacena el token devuelto en el almacenamiento local para validar futuras solicitudes al *backend*.

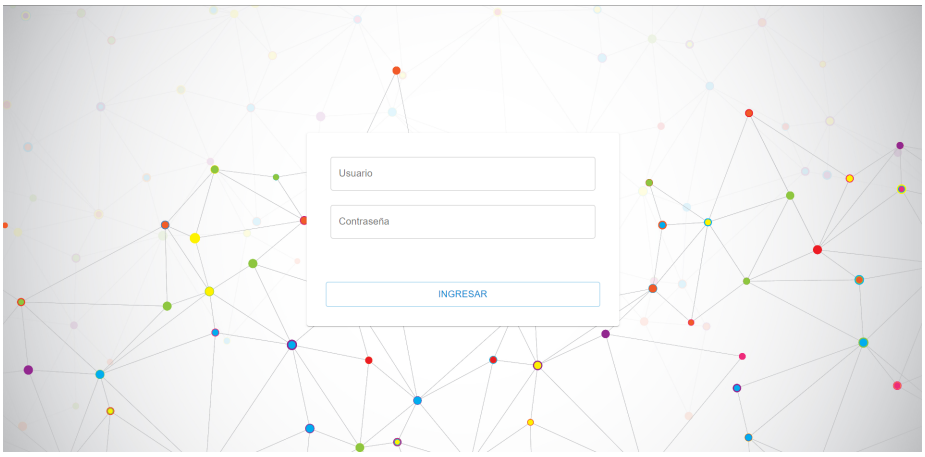


Figura 8.1: Formulario de inicio de sesión del sitio.

La pantalla del Chat es simple (Figura 8.2) y su propósito es permitir al usuario elegir entre tres versiones diferentes: la primera utiliza Regex, la segunda utiliza la similitud de coseno con TF-IDF, y la tercera utiliza un modelo de *machine learning* entrenado con datos de frases. El Chat es un componente de React llamado Chatbot, de la librería *react-chatbot-kit*. El selector de versiones modifica la ruta del *backend* para enviar y recibir mensajes según la versión seleccionada.

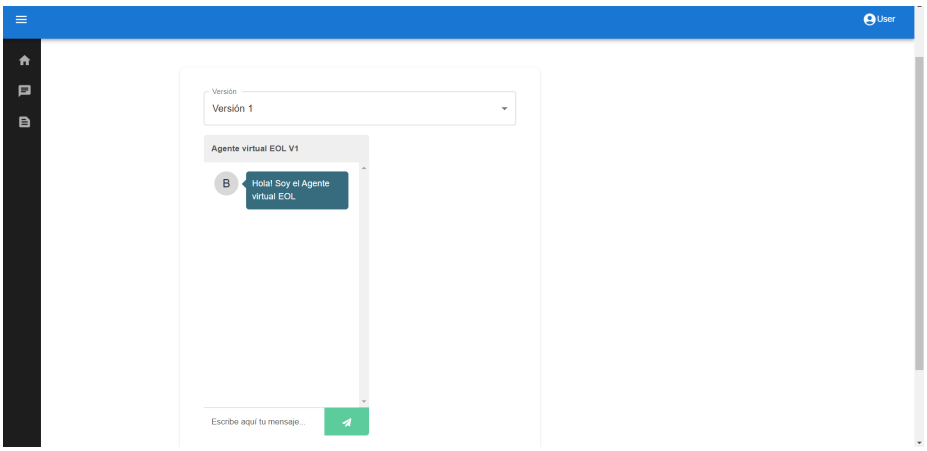


Figura 8.2: Página principal del sitio.

Una vez elegida la versión del chat con la que se quiere interactuar basta enviar un mensaje para que el sistema retorne la respuesta de la clase.

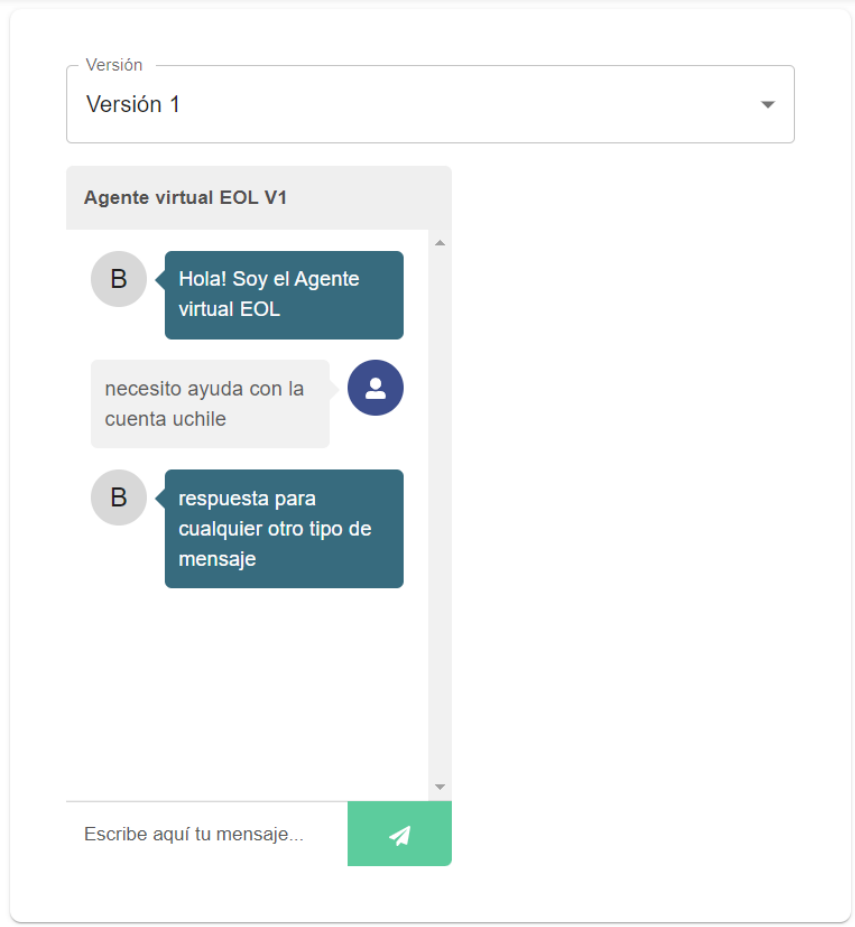


Figura 8.3: Interacción con la versión 1.

Como este frontend se encuentra desarrollado en **React** permite el cambio asíncrono de

componentes, de esta forma la respuesta puede probarse con las distintas versiones para comprobar la respuesta de cada versión y compararla con las demás.

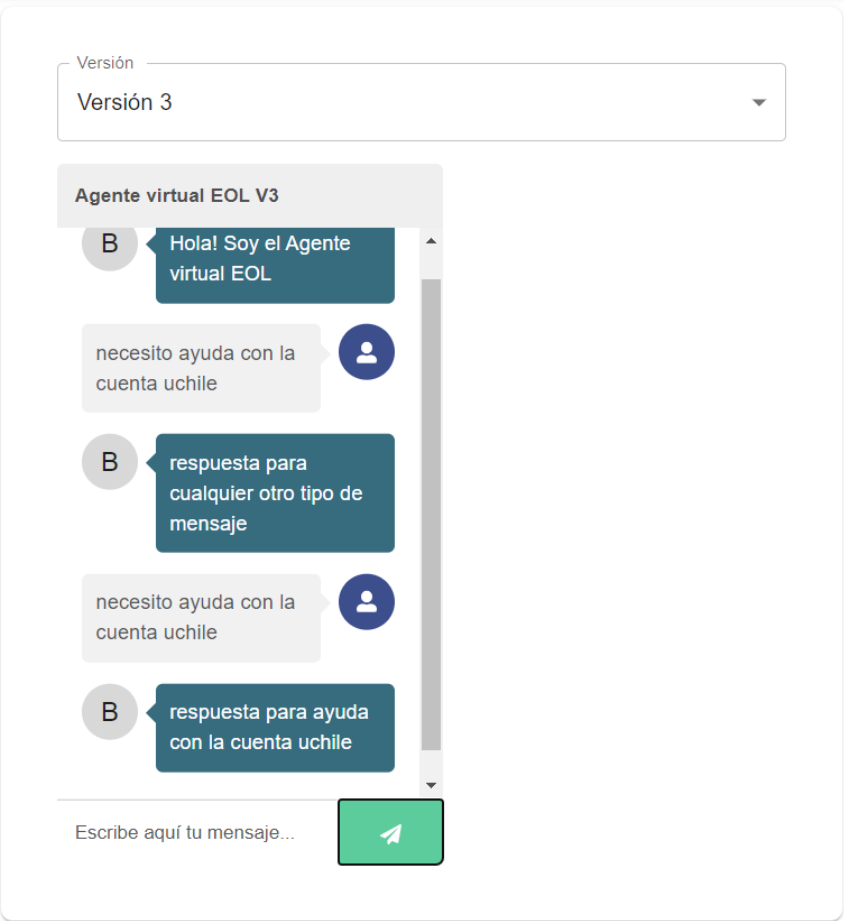


Figura 8.4: Interacción con la versión 2.

Si bien la comunicación no es fluida por el momento en cada interacción de usuario se clasifica la entrada para responder con la respuesta a la clase correspondiente.

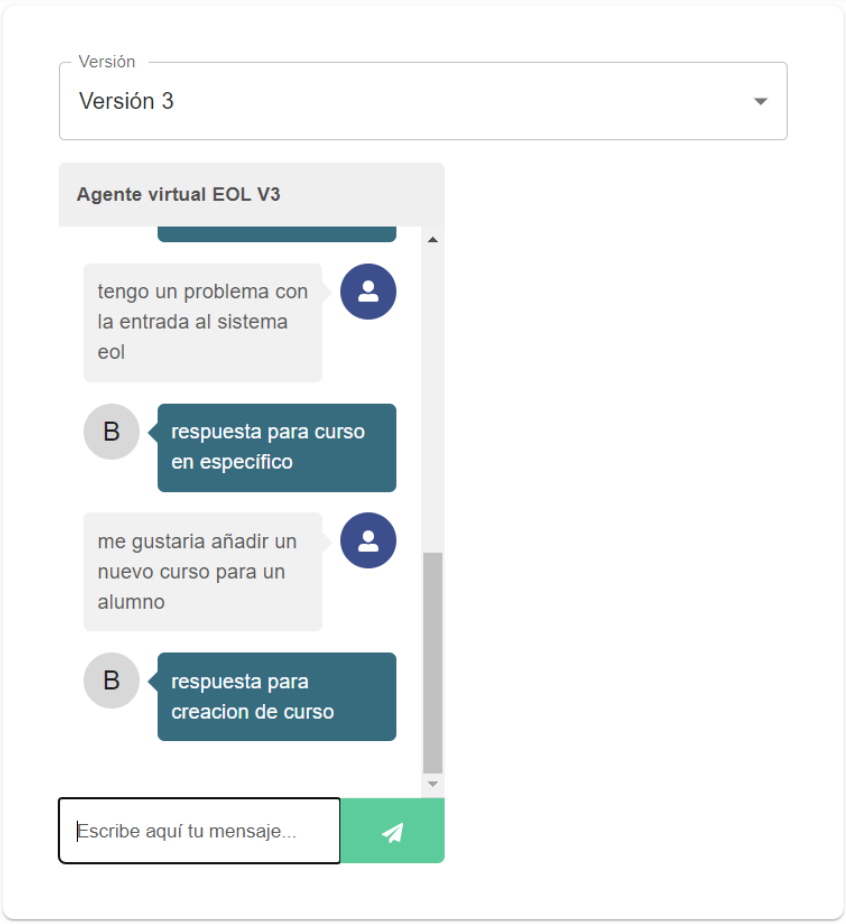


Figura 8.5: Interacción con la versión 3.

La vista del mantenedor de frases se presenta en forma de tabla (Figura 8.4), permitiendo visualizar y crear nuevas frases. Además, este componente facilita la integración de futuras funcionalidades como la edición y eliminación de frases existentes en el sistema.

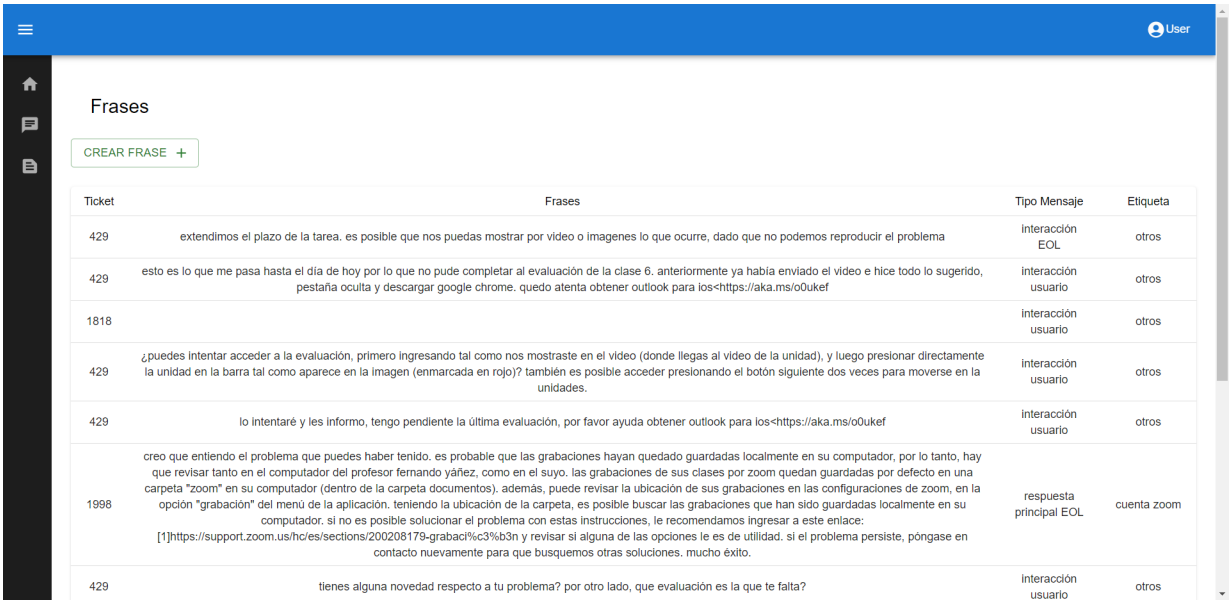


Figura 8.6: Mantenedor de frases del sitio.

# Capítulo 9

# Evaluación de los resultados

## 9.1. Preprocesamiento de datos

El vocabulario inicial generado sin preprocesamiento, que incluía todos los tipos de mensaje y no consideraba las clases con dos casos o menos (ver Tabla 9.1), constaba de 9787 palabras, muchas de las cuales no tenían sentido debido al formato original de los datos (correos electrónicos) y al procesamiento inicial deficiente, dejando correos, nombres o ruts en el texto. Esto era perjudicial para el entrenamiento del modelo, aunque podía resultar útil para su validación. Inicialmente, sin eliminar estas palabras innecesarias, se obtenía una *accuracy* del 50 % en los sets de validación y test.

Para reducir este vocabulario, se eliminaron las stopwords y se aplicó *stemming* mediante el *stemmer* de NLTK en español, lo que redujo el número de palabras únicas a 6451. Sin embargo, aún quedaban nombres, apellidos y símbolos en los textos. Al eliminar estos datos, el vocabulario final se redujo a 4109 palabras.

### 9.1.1. Separación de los datos

Antes de entrenar el modelo, se llevó a cabo la separación de los datos en tres conjuntos: entrenamiento, validación y prueba. Se utilizó una técnica llamada muestreo estratificado con asignación proporcional, que consiste en una representación estadística en la que se busca mostrar el total de las etiquetas de acuerdo a su ponderación en el conjunto total de frases. En este caso, se cuenta con un universo de 1081 frases que se distribuyen en 13 clases (Tabla 9.1).

Tabla 9.1: Etiquetas y la cantidad de apariciones.

| Etiqueta | Cantidad |
|---|---|
| activacion de cuenta | 38 |
| ayuda con certificados | 115 |
| cambio de correo | 30 |
| creacion de curso | 115 |
| creacion de usuario | 44 |
| contraseña | 44 |
| cuenta uchile | 178 |
| cuenta zoom | 129 |
| cuenta bloqueada | 2 |
| inicio de sesión | 0 |
| curso en específico | 357 |
| inasistencia | 1 |
| pagos | 28 |

Existen muchos casos etiquetados como "otros", pero, en general, no se les reconoció algún tipo de mensaje principal. En total, hay 2108 *tickets* que se marcaron con esta etiqueta, de los cuales 52 tienen un mensaje donde se puede reconocer una "pregunta principal del usuario" sin embargo siguen perteneciendo al tipo otros debido a lo ambiguo del contenido. Se decidió omitir el resto de los casos para no dedicar tiempo a los *tickets* que se encontraban fuera del ámbito del proyecto, con el fin de ahorrar tiempo en el etiquetado.

Con la base de datos cargada, se separaron los conjuntos de datos según la proporción 70 % Entrenamiento, 20 % Validación y 10 % Test.

## 9.2. Entrenamiento y evaluación

Todos los gráficos que se muestran a continuación en este capítulo corresponden a los resultados de *accuracy* y *loss* entre el set de validación y entrenamiento durante los 200 *epochs* y las tablas los resultados de todas las métricas en el set de *test* incluyendo las anteriores y *Precision*, *Recall*, *F1-Score* .

### 9.2.1. Primer modelo: *One-hot encoding* usando *stemming*

Se inició el entrenamiento con los conjuntos de datos creados con el objetivo de encontrar el modelo óptimo para los datos. El modelo inicial estaba compuesto por tres capas con 128, 64 y 10 neuronas (correspondientes a las clases de salida, descontando las que tienen dos casos o menos). Sin embargo, la *accuracy* obtenida tanto en el conjunto de prueba como en el conjunto de evaluación no superó el 50 % y el de test se mantuvo bajo el 48 %(Figura 9.2).

Si bien estos resultados son poco alentadores, este primer acercamiento sirve como punto de partida para mejorar los resultados.

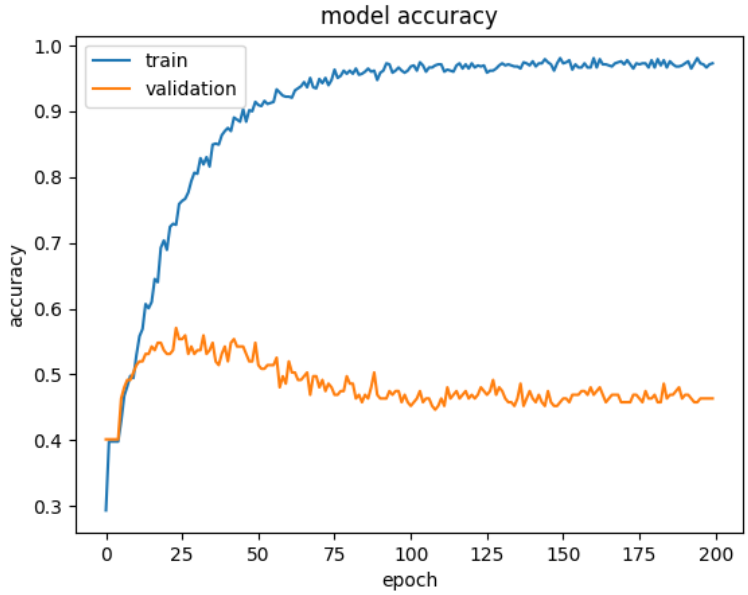


Figura 9.1: Accuracy obtenida usando stemming.

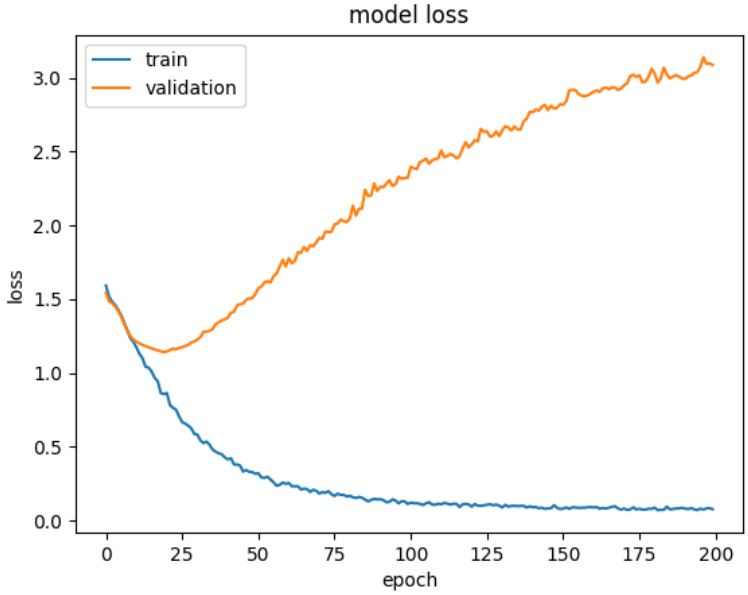


Figura 9.2: Loss obtenido usando stemming.

Tabla 9.2: Métricas obtenidas primer modelo.

| Accuracy | Loss | F1-Score | Precision | Recall |
|---|---|---|---|---|
| 0.5294 | 3.0379 | 0.5397 | 0.5862 | 0.5000 |

### 9.2.2. Segundo modelo: *One-hot encoding* sin uso de *stemming*

En la segunda iteración se probó la red neuronal sin el uso de stemming, con el fin de evaluar cuánto afectaba al rendimiento del modelo la decisión de utilizar o no este método de procesamiento de texto. En las Figuras 9.4 y 9.5 se observa la *accuracy* y la *loss* del modelo inicial sin stemming. Los resultados del set de *test* fueron de 47 % en promedio.

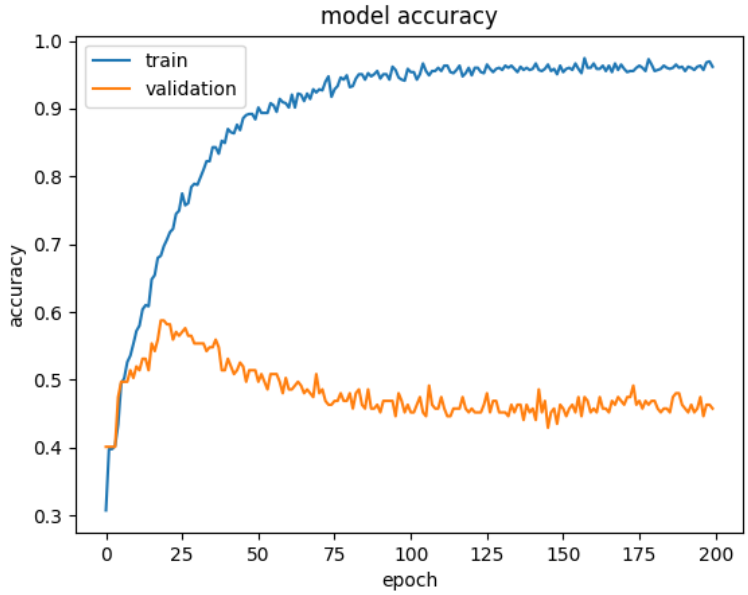


Figura 9.3: Resultados sin uso de stemming.

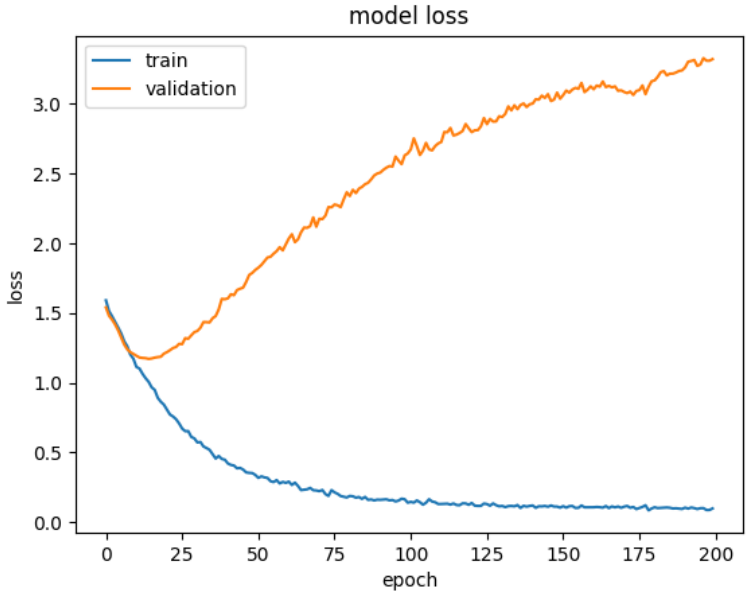


Figura 9.4: Loss sin uso de Stemming.

En este caso, se puede observar que no hubo una diferencia significativa en la *accuracy* del modelo al eliminar el uso de stemming. Sin embargo, la variación de *accuracy* entre cada *epoch* se volvió más notoria, y la curva de aprendizaje se ve menos suavizada, por otro lado hay un notorio efecto en el set de *test* donde la variación entre *Precision* y *Recall* genera un *F1-Score* muy bajo.

Tabla 9.3: Métricas obtenidas segundo modelo.

| Accuracy | Loss | F1-Score | Precision | Recall |
|---|---|---|---|---|
| 0.4704 | 3.1827 | 0.1091 | 0.7500 | 0.0588 |

### 9.2.3. Tercer modelo: Reducción de clases, *One-hot encoding* usando *stemming*

Debido a la gran disparidad en la cantidad de frases entre las etiquetas, se decidió probar el modelo con stemming utilizando una cantidad reducida de clases. En este caso, se utilizaron únicamente las cinco clases más comunes, las cuales se observan en la Tabla 9.2:

Tabla 9.4: Etiquetas más comunes.

| **Etiqueta** | **Cantidad** |
|---|---|
| ayuda con certificados | 115 |
| creacion de curso | 115 |
| cuenta uchile | 178 |
| cuenta zoom | 129 |
| curso en específico | 357 |

Los resultados de esta versión muestran una evidente mejora en comparación con la versión anterior (Figuras 9.5 y 9.6). Sin embargo, la *accuracy* en el conjunto de entrenamiento se acerca tanto al 100 % que parece ser una recta y notando la diferencia con la curva de validación es un indicador de que este modelo y los anteriores pueden estar sobreajustados. Los resultados del set de *test* fueron de 72 % en promedio en *accuracy*.

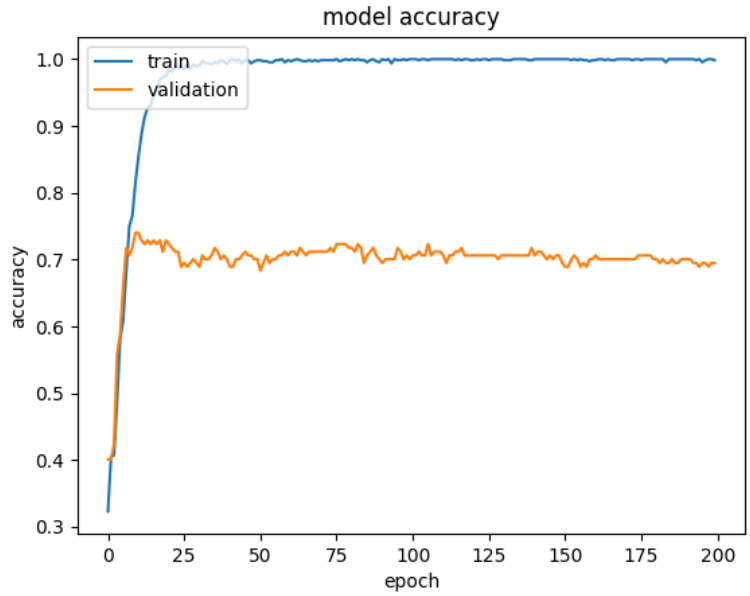


Figura 9.5: *accuracy* utilizando las cinco clases más comunes, mediante stemming.

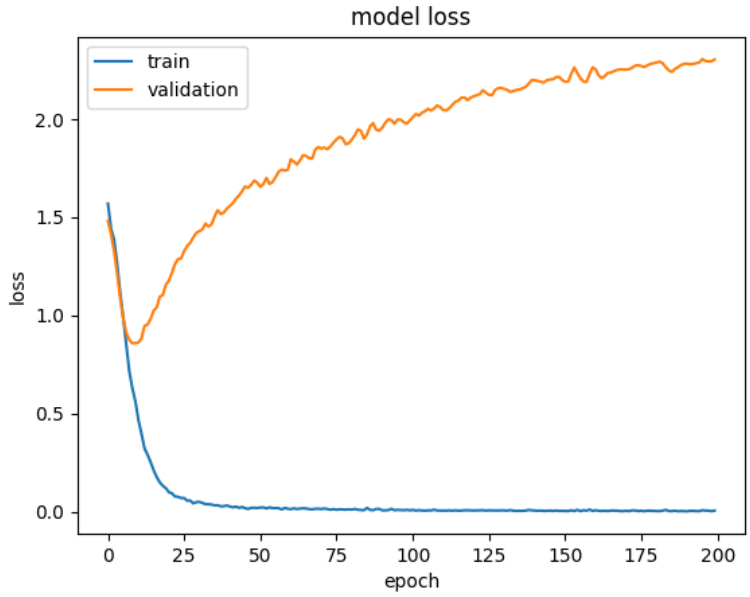


Figura 9.6: Loss utilizando las cinco clases más comunes, mediante stemming.

Tabla 9.5: Métricas obtenidas en el tercer modelo.

| Accuracy | Loss | F1-Score | Precision | Recall |
|---|---|---|---|---|
| 0.6928 | 2.2452 | 0.6433 | 0.6471 | 0.6395 |

### 9.2.4. Cuarto modelo: TF-IDF usando *stemming*

Se implementó TF-IDF para mejorar los resultados y reducir el sobreajuste. Esto se hizo para asignarle peso a las palabras del vocabulario, ya que en la implementación anterior, para cada frase el vector era binario (1 si la palabra aparecía y 0 en caso contrario), lo que no consideraba la relevancia de la palabra. En este caso, se utilizaron las mismas 10 clases que se emplearon en el modelo inicial. Los resultados se observan en las Figuras 9.7 y 9.8. Los resultados del set de *test* fueron 53 % de *accuracy* en promedio

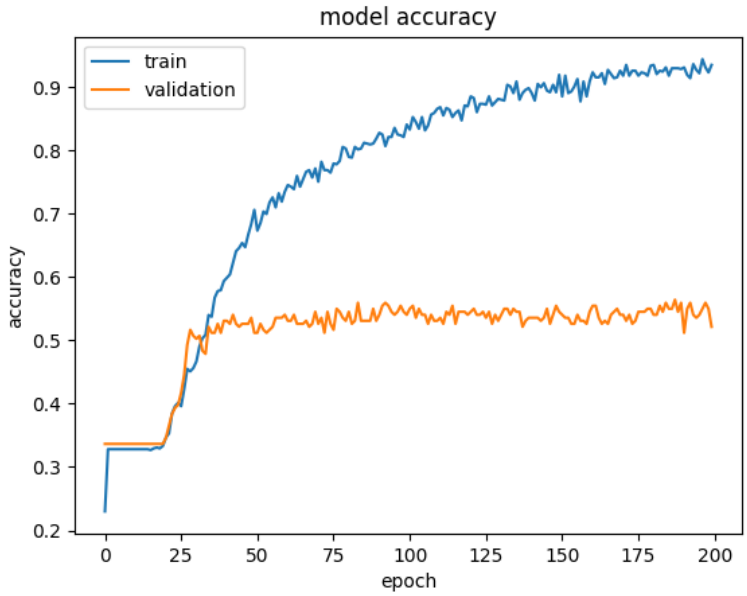


Figura 9.7: *accuracy* obtenida usando 10 clases .

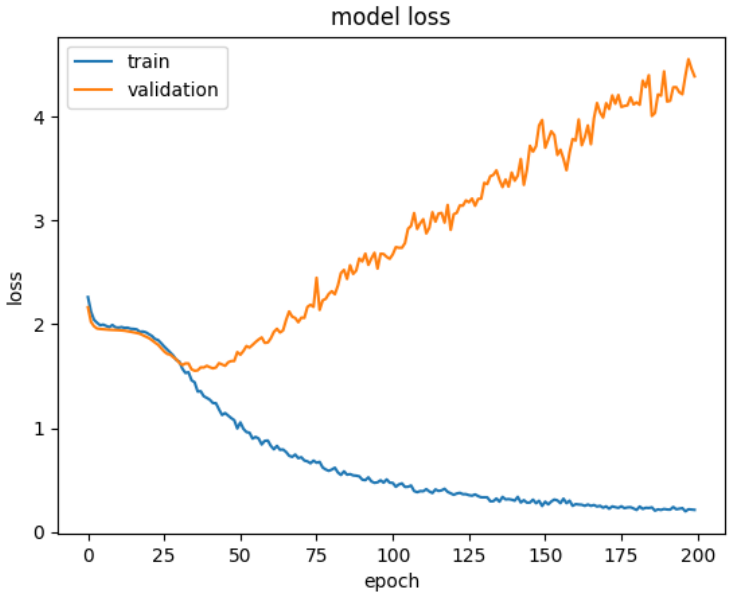


Figura 9.8: Loss obtenido usando 10 clases.

Al comparar estos resultados con los obtenidos en las Figuras 9.2 y 9.3, se puede notar una ligera mejora en la *accuracy*. En el conjunto de validación, la *accuracy* supera el 50 % al utilizar los mismos datos pero con un vocabulario determinado por los pesos de las palabras utilizando TF-IDF.

Tabla 9.6: Métricas obtenidas en el cuarto modelo.

| Accuracy | Loss | F1-Score | Precision | Recall |
|---|---|---|---|---|
| 0.5297 | 4.2740 | 0.5300 | 0.5324 | 0.5277 |

Aunque la *loss* observada es mayor para este modelo, la diferencia no es significativa. Además, es evidente la diferencia en la *accuracy* entre ambos modelos, sugiriendo que el modelo que utiliza TF-IDF para la representación de los datos es mejor que el que no lo hace, gracias a la utilización de vectores de pesos más significativos ya que pasa de ser binario como es *one-hot encoding* a valores que reflejan la relevancia de cada palabra.

### 9.2.5. Quinto modelo: Reducción de clases, TF-IDF usando *stemming*

Similarmente al modelo anterior, se procedió a reducir las clases, utilizando las etiquetas de la Tabla 9.2, para evaluar si se mejora la *accuracy* y se reduce el sobreajuste. Los resultados se observan en las Figuras 9.9 y 9.10. Los resultados del set de *test* fueron de 66 % en promedio de *accuracy*.

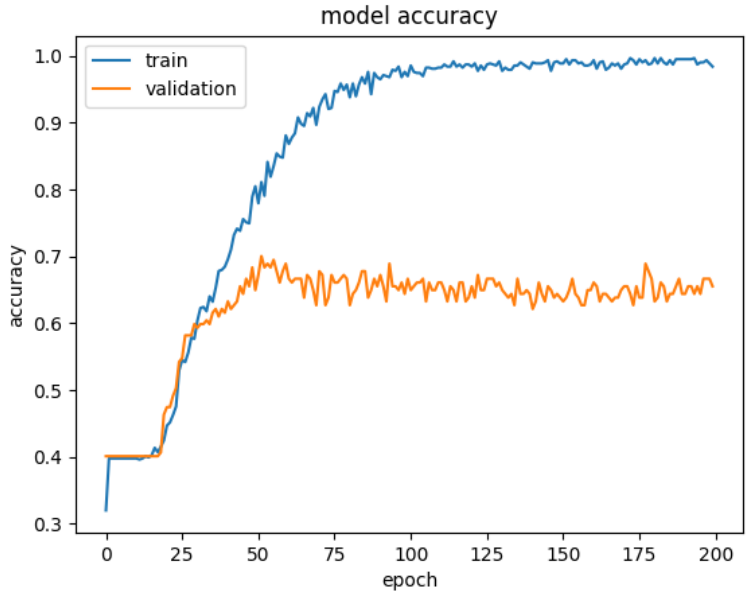


Figura 9.9: *accuracy* obtenida usando las cinco clases más comunes, mediante TF-IDF.

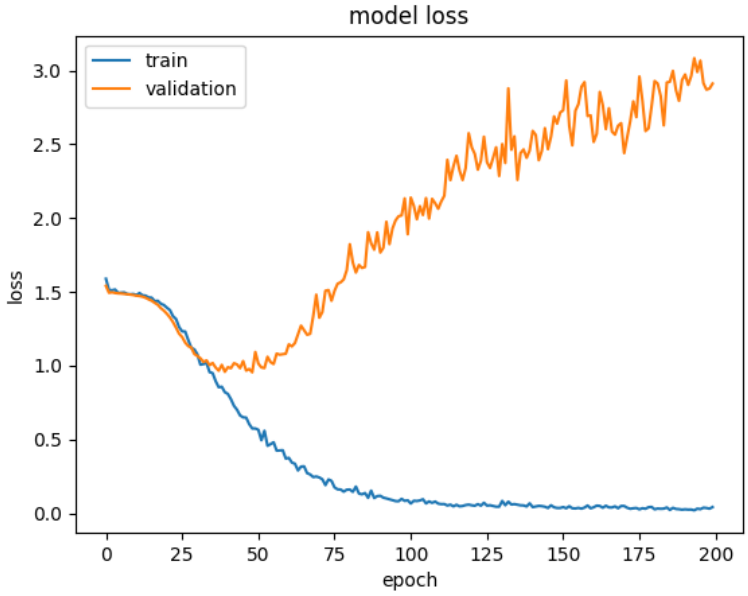


Figura 9.10: Loss obtenido usando las cinco clases más comunes, mediante TF-IDF.

Los resultados muestran una mejora en la *accuracy* en comparación con el modelo que utilizó TF-IDF con todas las clases, pero no supera la *accuracy* del conjunto de validación del modelo inicial con cinco clases (Figura 9.5). Además, el modelo parece estar menos sobreajustado, aunque en los últimos *epochs* la *accuracy* en el conjunto de entrenamiento se acerca al 100 %.

La *loss* observada se comporta muy similar a la del modelo inicial con cinco clases (Figura 9.5).

Tabla 9.7: Métricas obtenidas en el quinto modelo.

| Accuracy | Loss | F1-Score | Precision | Recall |
|---|---|---|---|---|
| 0.6860 | 3.1326 | 0.7018 | 0.7059 | 0.6977 |

### 9.2.6. Sexto modelo: Ejemplo de sobreajuste por sobremuestreo

Se decidió evaluar el desempeño de los modelos utilizando más datos, mediante la técnica de sobremuestreo, duplicando artificialmente los datos en aquellas clases con menos casos. Sin embargo, es importante aplicar esta técnica correctamente y solo en el set de entrenamiento, para evitar forzar la validación con datos que también se usaron para entrenar.

Aunque no se obtuvieron mejoras notables al aplicar correctamente la metodología mencionada anteriormente, es importante mencionar cómo cambian los resultados al aplicarla incorrectamente. En las Figuras 9.11 y 9.12 se muestran los resultados obtenidos al duplicar las clases antes de la división de los conjuntos, lo cual lleva a una falsa creencia de que se está mejorando el modelo, con una *accuracy* superior al 80 % en los conjuntos de validación y prueba. Es importante destacar que estos resultados son incorrectos, ya que se está validando y probando con datos que se utilizaron previamente para el entrenamiento (el mismo dato que fue duplicado, y utilizado para entrenar, se encuentra en los otros conjuntos).

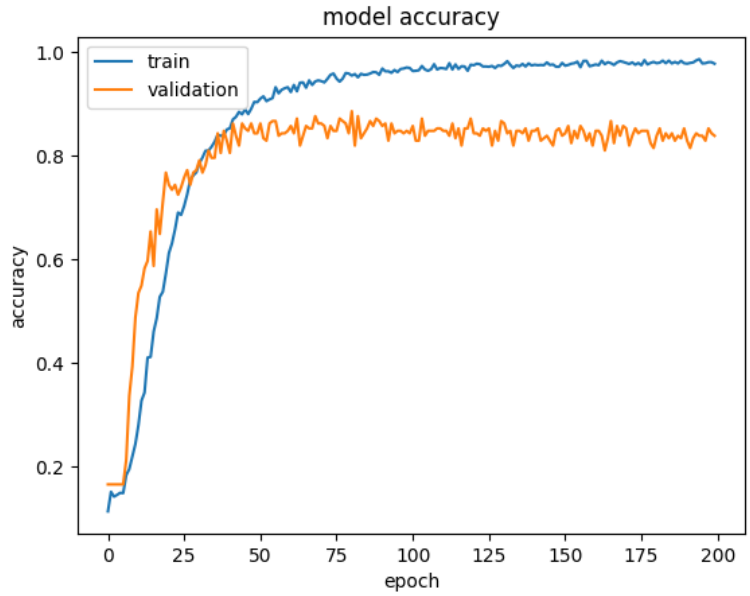


Figura 9.11: *accuracy* obtenida usando las 10 clases balanceadas, mediante TF-IDF.

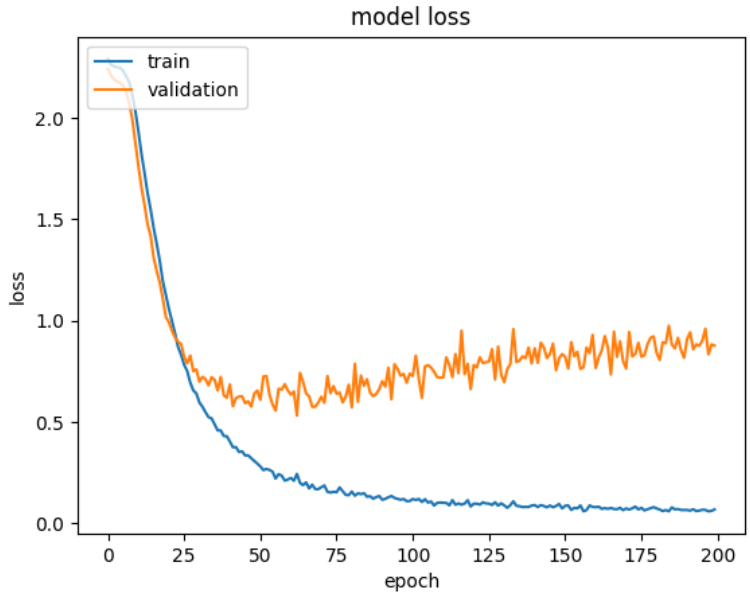


Figura 9.12: *Loss* obtenido usando las 10 clases balanceadas, mediante TF-IDF.

La duplicación de frases en las clases con menor densidad de datos también parece mejorar la *loss* del modelo, pero es importante tener en cuenta que esta es una estrategia incorrecta. Si no se reconoce esto, se podría pensar que este es un buen modelo. Por lo tanto, para este trabajo, se propone considerar el modelo que utiliza TF-IDF con las cinco clases más comunes (Figura 9.10) como el mejor, basado en los datos de entrenamiento validación y test.

Tabla 9.8: Métricas obtenidas en el sexto modelo.

| Accuracy | Loss | F1-Score | Precision | Recall |
|---|---|---|---|---|
| 0.8043 | 0.9321 | 0.7870 | 0.8004 | 0.7742 |

### 9.2.7. Resumen de modelos

Tabla 9.9: Resumen de métricas obtenidas en los distintos modelos.

| Modelo | Accuracy | Loss | F1-Score | Precision | Recall |
|---|---|---|---|---|---|
| 1 | 0.5294 | 3.0379 | 0.5397 | 0.5862 | 0.5000 |
| 2 | 0.4704 | 3.1827 | 0.1091 | 0.7500 | 0.0588 |
| 3 | 0.6928 | 2.2452 | 0.6433 | 0.6471 | 0.6395 |
| 4 | 0.5297 | 4.2740 | 0.5300 | 0.5324 | 0.5277 |
| **5** | **0.6860** | **3.1326** | **0.7018** | **0.7059** | **0.6977** |

Sin considerar el modelo balanceado como el mejor debido a la falsa impresión de calidad debido a la duplicación de registros en los sets de validación entrenamiento y test, esto se puede entender como que el modelo no aprendió, simplemente memorizó. El quinto modelo tiene las mejores métricas de *accuracy* y *F1-Score*, se puede considerar un mejor modelo en comparación con los demás y para comprobar que la distancia entre las curvas de entrenamiento y validación no signifique un *overfitting* se comprobó con la técnica de validación cruzada.

### 9.2.8. Validación cruzada

La validación cruzada selecciona aleatoriamente los datos de entrenamiento y prueba en varias iteraciones, en este caso se escogieron 10, calculando las métricas del modelo en cada una de ellas. De esta manera, se puede probar que el modelo funciona bien independientemente de la partición de los datos utilizados en cada iteración. En las Figuras 9.13 y 9.14 se muestran los resultados de la validación cruzada llevada a cabo al modelo que utiliza TF-IDF y que fue entrenado con las cinco clases más comunes.

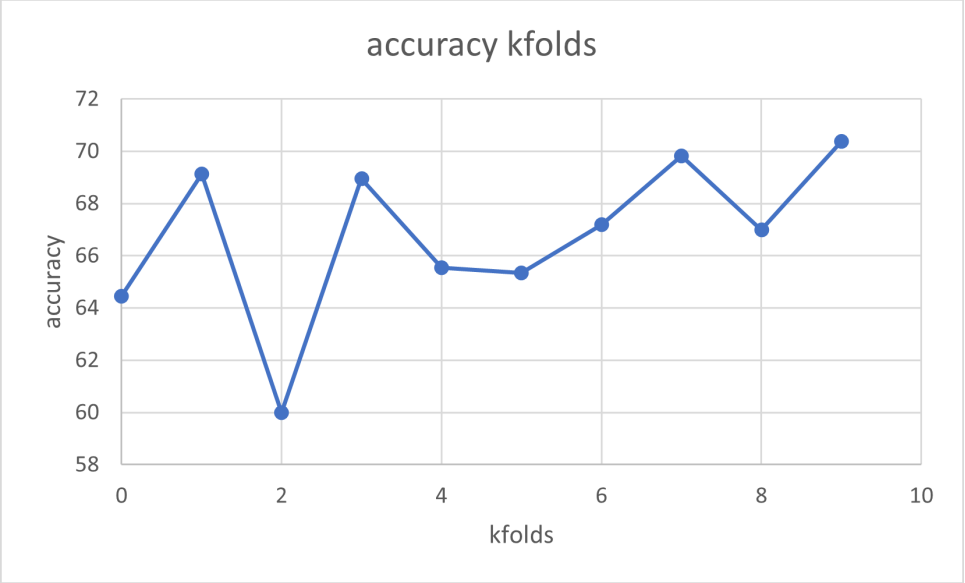


Figura 9.13: *accuracy* obtenida con el modelo que utiliza TF-IDF y las cinco clases más comunes en 10 iteraciones mediante validación cruzada.

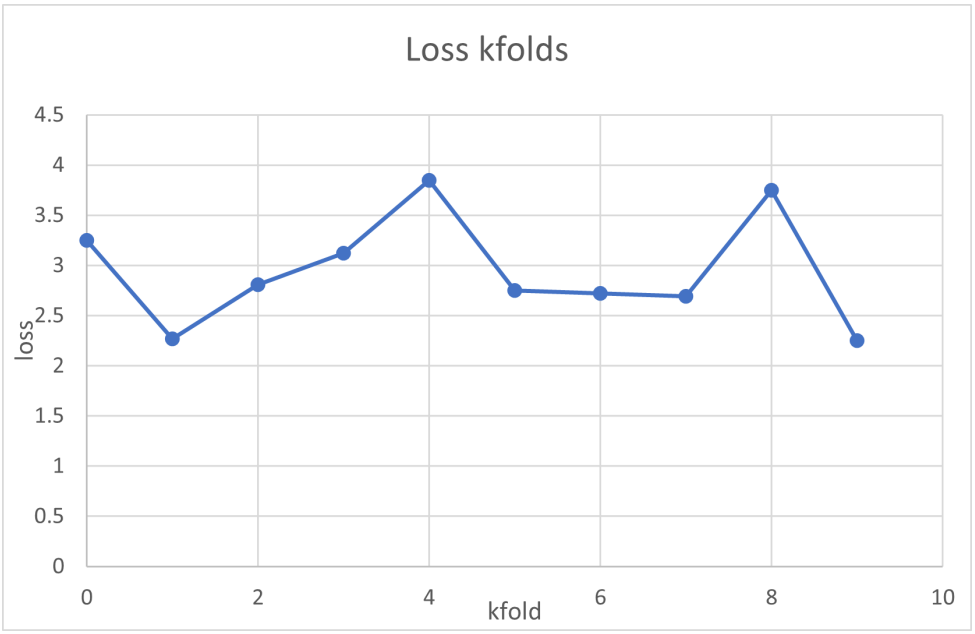


Figura 9.14: *Loss* obtenida con el modelo que utiliza TF-IDF y las cinco clases más comunes en 10 iteraciones mediante validación cruzada.

En promedio, se logró una *accuracy* del 66.8 % mediante la validación cruzada, lo que indica que dos tercios de las solicitudes podrían ser clasificadas correctamente si se limita el sistema a cinco tipos de clases.

La *loss* obtenida en la validación cruzada coincide con la *loss* observada en la Figura 9.10, lo que confirma que se trata del mismo modelo y que su comportamiento es consistente en distintas iteraciones de entrenamiento.

Tabla 9.10: Métricas obtenidas en cada *k-fold* de validación cruzada.

| Kfold | Accuracy | Loss | F1-Score | Precision | Recall |
|---|---|---|---|---|---|
| 0 | 0.6435 | 3.2670 | 0.6433 | 0.6471 | 0.6395 |
| 1 | 0.6914 | 2.3109 | 0.6944 | 0.7012 | 0.6879 |
| 2 | 0.6021 | 2.8041 | 0.6096 | 0.6115 | 0.6078 |
| 3 | 0.6923 | 3.1787 | 0.7004 | 0.6987 | 0.7022 |
| 4 | 0.6592 | 3.8022 | 0.6601 | 0.6607 | 0.6596 |
| 5 | 0.6575 | 2.6018 | 0.6550 | 0.6588 | 0.6512 |
| 6 | 0.6733 | 2.5582 | 0.6847 | 0.6758 | 0.6754 |
| 7 | 0.6989 | 2.4036 | 0.6860 | 0.6860 | 0.6860 |
| 8 | 0.6650 | 3.7897 | 0.6706 | 0.6786 | 0.6628 |
| 9 | 0.7047 | 2.3066 | 0.7085 | 0.7186 | 0.6987 |

# Capítulo 10

# Conclusión

El objetivo de este trabajo fue desarrollar un software de Inteligencia Artificial (IA) que facilite la resolución de tickets de los usuarios que acceden a la mesa de ayuda de EOL. De esta manera, podrían reducir la carga de trabajo de los responsables de la mesa de ayuda. La necesidad de este sistema surgió debido a la gran cantidad de *tickets* que se resuelven con respuestas simples y comunes, lo que resulta en horas de trabajo que podrían dedicarse a resolver los *tickets* más difíciles y que requieren una mayor intervención.

La solución se enfocó en tres aspectos: la etiquetación de los datos, la creación de una API REST, y el desarrollo y evaluación de un modelo que utilizara y predijera clases con los datos etiquetados inicialmente.

Estas soluciones lograron cumplir los objetivos a diferentes niveles. El etiquetado de los datos fue satisfactorio y se utilizó el etiquetador sin problemas por parte de terceros. Estos datos comprenden 13.153 mensajes de un total de 3.854 tickets que se clasificaron entre 13 etiquetas estos mensajes son los que se utilizaron para entrenar un modelo de inteligencia artificial, aunque inicialmente eran 14 etiquetas, sin embargo una de estas clases quedó vacía luego de que en el etiquetado no se encontrara ningún mensaje principal de usuario. .

La API como solución es muy efectiva, ya que cumple el objetivo de desarrollar un agente virtual adaptable para EOL. Esta API permite proporcionar respuestas automáticas basadas en un modelo entrenado. Con los datos actuales, el modelo logra 70 % de *accuracy*, lo que significa que EOL puede resolver satisfactoriamente más de 2 de cada 3 preguntas que llegan al sistema. Sin embargo, queda trabajo por hacer, y es aquí donde el backend resulta beneficioso, ya que permite integrar nuevas frases, etiquetas y tipos de mensajes, o eliminarlos. De esta manera, EOL tiene control sobre el reentrenamiento del modelo con los nuevos datos que ingresen al sistema.

La evaluación del modelo se centró en probar distintas configuraciones para la red neuronal, se probó con distintos ajustes combinando varios de ellos, usando distintas funciones de activación, cantidades de neuronas, capas, optimizadores y metodologías de preprocesamiento (stopwords, stemming, lematización y TF-IDF) la mayoría de los ajustes no tenían un gran impacto, sin embargo en este trabajo se explicaron los resultados obtenidos de los más importantes que tenían resultados con métricas llamativas y variables. Para aprobar

el último modelo se usó validación cruzada, de ese modo comprobar que los resultados de precisión no estaban siendo afectados por las particiones de los sets. Si bien los resultados de precisión tanto en los set de validación y entrenamiento no fueron muy altos asegurando únicamente un rendimiento de entre 66 % a 71 % de *accuracy*, esto tiene explicación, los datos fueron etiquetados por externos a EOL y puede en muchos casos haber ocurrido una falta de entendimiento al problema, esto significa en clases con *features* poco diferenciadoras y por tanto un no muy alto valor de precisión en clasificaciones de datos nuevos. Otro factor importante es la densidad de los datos, muchos de los *tickets* recibidos durante el año eran pruebas de funcionamiento de la plataforma, para validar que la mesa de ayuda funcionaba correctamente o interacciones con plataformas externas que enviaban correos en inglés, esto significó que una gran cantidad de *tickets* fueran etiquetados como **otros** y no sirvieran para el entrenamiento del modelo. La falta de datos también resulta en un sobreajuste constante y en una baja precisión de validación y test. Pese a esto el sistema permitirá a EOL administrar estos datos para que puedan etiquetar nuevos tickets o volver a etiquetar los datos con que fue entrenado actualmente el sistema, de esta forma les permite internar el proceso para mejorarlo en base a sus conocimientos de los temas relacionados a su empresa.

Para EOL, esta solución resulta útil desde una perspectiva experimental y práctica. Aunque se trata de una versión beta, pueden implementarla en sus sistemas, mejorarla y modificarla a su gusto. Además, pueden incluir a futuros investigadores para que extiendan, mejoren y acerquen este proyecto a su versión productiva.

## 10.1. Mejoras a futuro

Dado el alcance y complejidad del proyecto, todavía hay trabajo por hacer y espacio de mejora. Por lo tanto, esta memoria se considera una versión base y de prueba, destinada a ser utilizada por futuros investigadores interesados en esta área. A continuación, se presenta una lista de las mejoras más importantes que deben abordarse en el futuro:

- Mejorar el manejo de *tickets*, de tal forma que la comunicación genere automáticamente un *ticket* en el sistema y se le asigne un estado de resuelto o no resuelto.
- Aplicar mejoras en el *frontend*, ya sea mediante la mejora del *frontend* actual o la extensión del *frontend* de EOL para que consuma la API.
- Mejorar la comunicación para que sea más fluida, lo cual se puede lograr identificando los mensajes de saludos e identificación como una etiqueta más o como un tipo de mensaje. Esto requiere un trabajo de etiquetado manual que, lamentablemente, no se abordó en este proyecto, esto significa extender los tipos de mensaje de 4 a 6 incluyendo **saludo de usuario** y **saludo de EOL**.
- Estudiar la generación de respuestas automáticas en base a las respuestas enviadas por EOL, esto es complicado con el formato actual de los textos, hay soluciones no triviales para algunos temas que deberían quedar fuera, sin embargo, sería una buena idea estudiar esta posibilidad.

- Conectar el etiquetador con la base de datos y backend para optimizar trabajo, el etiquetador como fue la primera parte del trabajo, se llevó a cabo usando csv para ahorrar el tiempo que tomaba crear un modelo y leva bases de datos. Sería una buena idea que este etiquetador se conectara directamente a la base de datos.
- Permitir la carga masiva de frases al sistema, ya sea mediante csv o json, etc. Que permita que se puedan incluir múltiples frases nuevas al modelo de datos.

# Bibliografía


[1] chatbot — definition of chatbot in english by lexico dictionaries. `https://www.dictionary.com/browse/chatbot`.

[2] Heroku. `www.heroku.com`.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Eleni Adamopoulou and Lefteris Moussiades. An overview of chatbot technology. In Ilias Maglogiannis, Lazaros Iliadis, and Elias Pimenidis, editors, *Artificial Intelligence Applications and Innovations*, pages 373–383, Cham, 2020. Springer International Publishing.

[5] Baidaa Alsafy, Zahoor Mosad, and Wamidh Mutlag. Multiclass classification methods: A review. 12 2020.

[6] Daniel Berrar. *Cross-Validation*. 01 2018.

[7] Menal Dahiya. A tool of conversation: Chatbot. *International Journal of Computer Sciences and Engineering*, 5(5):158–161, 2017.

[8] Christopher Fox. A stop list for general text. *SIGIR Forum*, 24(1–2):19–21, sep 1989.

[9] Douglas M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004. PMID: 14741005.

[10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[11] Yang Liu, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. Topical word embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Feb. 2015.

[12] Panitan Muangkammuen, Narong Intiruk, and Kanda Runapongsa Saikaew. Automated thai-faq chatbot using rnn-lstm. In *2018 22nd International Computer Science and Engineering Conference (ICSEC)*, pages 1–4. IEEE, 2018.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[15] David Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation. *Mach. Learn. Technol.*, 2, 01 2008.

[16] Shahzad Qaiser and Ramsha Ali. Text mining: Use of tf-idf to examine the relevance of words to documents. *International Journal of Computer Applications*, 181, 07 2018.

[17] Bayan Abu Shawar and Eric Atwell. Chatbots: are they really useful? In *Ldv forum*, volume 22, pages 29–49, 2007.