Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

Detección de depresión a partir de comentarios escritos en español usando Procesamiento del Lenguaje Natural

Alumno: Raúl Moreno Sánchez

Tutor: Salud María Jiménez Zafra

Dpto: Informática

UNIVERSIDAD DE JAÉN

Dª Salud María Jiménez Zafra tutora del Trabajo Fin de Grado titulado: **Detección de depresión a partir de comentarios escritos en español usando Procesamiento del Lenguaje Natural**, que presenta Raúl Moreno Sánchez, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, febrero de 2024

El estudiante	El tutor
MORENO SANCHEZ RAUL - 77434741C Firmado digitalmente por MORENO SANCHEZ RAUL - 77434741C Fecha: 2024.02.20 20:03:38 +01'00'	JIMENEZ ZAFRA SALUD MARIA - 77370897R Firmado digitalmente por JIMENEZ ZAFRA SALUD MARIA - 77370897R Fecha: 2024.02.20 20:23:57 +01'00'
Raúl Moreno Sánchez	Salud María Jiménez Zafra

Agradecimientos

Quisiera expresar mi sincero agradecimiento a todas las personas que contribuyeron de diversas maneras a la realización de este Trabajo Fin de Grado.

En primer lugar, quiero agradecer a mi tutora del TFG, Salud María Jiménez Zafra, por su orientación, apoyo y dedicación a lo largo de todo este proceso. Sus conocimientos y su guía fueron fundamentales para alcanzar los objetivos de este trabajo.

Agredecer también a Alberto Fernández Hernández, un miembro del equipo TextualTherapists con el que he participado en la tarea de MentalRiskES. Sin su conocimiento del código con el que he partido, no hubiese sido capaz de realizar este trabajo.

No puedo dejar de mencionar el respaldo recibido por parte de mi familia y mi pareja. Gracias a su constante apoyo emocional y por creer en mí en cada paso de este viaje educativo.

Este Trabajo Fin de Grado representa el resultado de esfuerzo, dedicación y colaboración de muchas personas, a quienes estoy profundamente agradecido.

¡Muchas gracias a todos!

Tabla de contenidos

Lista de figuras

Lista de tablas

Lista de listados de código

Capítulo 1

INTRODUCCIÓN

Según el informe mundial sobre salud mental de la Organización Mundial de la Salud (OMS) [1], 1 de cada 8 personas en el mundo sufre algún trastorno mental. La pandemia de la COVID-19 ha elevado la prevalencia de la ansiedad y la depresión a más del 26 % en sólo el primer año de la pandemia. La OMS considera que la identificación precoz es una intervención eficaz clave para prevenir estos problemas. En consecuencia, existe un interés creciente por detectar e identificar trastornos mentales en los flujos de las redes sociales.

Con este Trabajo Fin de Grado, se pretende aprovechar los avances en el campo del Procesamiento del Lenguaje Natural (PLN) para abordar un gran problema existente en la actualidad, la detección de depresión. Para ello, se va a crear un sistema de clasificación automática para la detección de depresión a partir de comentarios escritos en español usando PLN. En concreto, se va a desarrollar un sistema para participar en la Tarea 2.a: "Detección de depresión"de MentalRiskES [2], una novedosa tarea para la identificación temprana de riesgo de trastornos mentales en comentarios en español de usuarios de Telegram, la cual se ha organizado en el marco de la campaña de evaluación IberLEF 2023 [3]. Aunque la tarea es sobre comentarios en español, el sistema que presentamos es en inglés y aplica una traducción de los textos español-inglés, por lo que se va a analizar dicho sistema y se va a realizar una modificación de las variables del modelo al español para usar los textos directamente en español y realizar un análisis de resultados para ver cómo de **preciso** y **rápido** es el modelo con los cambios realizados

1.1. Motivación

La detección temprana y, sobre todo, precisa, de la depresión, es una importante tarea en el ámbito de la salud mental en la actualidad. La depresión es una enfermedad que afecta a millones de personas en todo el mundo. Esta enfermedad causa un gran deterioro en el bienestar y la calidad de vida de estas personas. Sin embargo, en muchos casos, las personas que padecen depresión no buscan ayuda o no son diagnosticadas de la forma adecuada debido a la falta de conciencia o dificultad para expresar sus emociones abiertamente.

Según la OMS, se estima que hay alrededor de 300 millones de personas en el mundo que sufren depresión. En Norte América la probabilidad de tener un episodio de depresión grave en el período de tiempo de un año es de un 3-5 % para hombres y un 8-10 % para mujeres [4].

En este contexto, el PLN aparece como una herramienta poderosa para ayudar a mejorar este problema. La enorme cantidad de texto que aparece en plataformas digitales, como redes sociales, blogs o foros, ofrece una gran fuente de información sobre el estado emocional y mental de las personas. Analizar estos textos de forma automática puede brindar una oportunidad para detectar señales de depresión, identificar patrones lingüísticos asociados y proporcionar una evaluación objetiva y lo más temprana posible a los pacientes.

El PLN se enfoca en el análisis de aspectos acústicos y lingüísticos del lenguaje humano derivados del texto y del habla y pueden ser integrados en sistemas de aprendizaje automático (“machine learning” en inglés) para clasificar la depresión y, si es posible, su nivel de gravedad.

El desarrollo de un sistema de detección de depresión puede tener un gran impacto en la atención y tratamiento de la salud mental. La detección temprana y precisa permitiría a los profesionales de la salud intervenir de manera oportuna, dando apoyo a aquellos que lo necesitan, incluso antes de que busquen ayuda activamente. Además, un enfoque automatizado puede brindar una evaluación objetiva basada en datos.

1.2. Propósito

Para poder realizar la correcta clasificación de depresión en usuarios, se necesita un sistema de clasificación. Este proyecto no se basa en realizar uno desde cero, si

no, en analizar y adaptar un sistema existente en inglés realizado por la ONG SoGoo-Data [5]. Este sistema es el que ha usado para participar en una de las tareas de MentalRiskES, concretamente la tarea 2.a que se basa en la clasificación binaria de detección de depresión, que se enmarca dentro de la campaña de evaluación IberLEF 2023.

También, se va a realizar una modificación de dicho sistema de clasificación binaria para poder usarlo con textos escritos en español. Los textos proporcionados por la campaña de evaluación son textos en español, pero, el sistema que se ha usado para clasificarlos trabaja con textos escritos en inglés, por lo que primero se realizara una traducción al inglés del conjunto de datos. Por esta razón, se quiere realizar una modificación de las variables del sistema para que admita los textos escritos en español. Para ello, se modificarán las variables necesarias y se realizarán pruebas con el mismo conjunto de datos proporcionado y se realizará un análisis de los resultados obtenidos para ver el desempeño del sistema con los cambios realizados.

1.3. Objetivos

A continuación, se van a definir los objetivos a seguir para la realización del proyecto:

1. Realizar un estudio o investigación del uso del Procesamiento del Lenguaje Natural en el ámbito de la detección de depresión.
2. Explicar qué son las campañas de evaluación y la tarea en la que se ha participado.
3. Analizar el conjunto de datos y el sistema de clasificación.
4. Evaluar el correcto funcionamiento del sistema de clasificación, haciendo uso de un análisis de errores y una matriz de confusión.
5. Implementar las modificaciones de las variables del sistema de clasificación.
6. Evaluar el correcto funcionamiento del sistema de clasificación con las variables modificadas.
7. Realizar una comparación de los resultados del sistema de clasificación, previo y posterior a la modificación de las variables.

8. Redactar una memoria que recoja todos los pasos realizados y todos los objetivos anteriores.

1.4. Resultados esperados

Como resultado de este Trabajo Fin de Grado, se espera llevar a cabo un análisis y evaluación del sistema de clasificación en inglés implementado por el equipo TextualTherapists [6], del cual formo parte, para participar en la tarea MentalRiskES de la campaña de evaluación IberLEF 2023. Sabiendo esto, se busca obtener una comprensión profunda de su funcionamiento, rendimiento y limitaciones.

Posteriormente, se realizarán las adaptaciones necesarias en el sistema de clasificación para que pueda procesar textos escritos en español. Esto implicará ajustes en las herramientas de PLN y la incorporación de recursos lingüísticos específicos del idioma español. Estas modificaciones permitirán cambiar la capacidad del sistema para analizar y clasificar textos escritos en español.

Una vez realizado el proceso de adaptación, se llevará a cabo la evaluación del sistema modificado. Este análisis de los resultados permitirá determinar la efectividad y las mejoras logradas en la detección de depresión en textos escritos en español.

Se espera que este Trabajo Fin de Grado proporcione una visión clara y detallada de las características y el rendimiento del sistema de clasificación original, así como de su adaptación al español. Además, se espera demostrar la viabilidad y utilidad de las técnicas de PLN en el contexto de la detección de depresión, lo cual podría tener un impacto positivo en la detección temprana y el tratamiento de este trastorno mental.

1.5. Planteamiento de tareas

A continuación, mostraré una tabla de las tareas que he seguido para la realización de este proyecto:

Como podemos ver en la Tabla 1.1, está indicado también, el tiempo dedicado a cada tarea y cuando se inició y se terminó.

Tarea	Duración	Inicio	Fin
Estudio de la depresión en medios sociales	10 horas	10 de junio de 2023	14 de junio de 2023
Estudio de las campañas de evaluación y sus tareas	15 horas	15 de junio de 2023	10 de julio de 2023
Análisis del sistema de clasificación de depresión	30 horas	2 de octubre de 2023	27 de octubre de 2023
Adaptación del sistema de clasificación para textos en español	95 horas	28 de octubre de 2023	20 de diciembre de 2023
Análisis del sistema de clasificación adaptado al español	20 horas	4 de enero de 2024	19 de enero de 2024
Generación de una memoria que recoja todo el proceso realizado	130 horas	14 de junio de 2023	19 de febrero de 2024

Tabla. 1.1: Planificación de las tareas

1.6. Estructura del proyecto

Finalmente, para terminar este primer apartado introductorio a este proyecto, se listarán las diferentes partes en las que se ha dividido el proyecto, detallando brevemente cada apartado:

- Investigación sobre el estado del arte del Procesamiento del Lenguaje Natural: definición y concepto del PLN. Descripción de lo qué es, su clasificación y las diferentes aplicaciones que se le pueden dar.
- Investigación sobre las campañas de evaluación: descripción de qué son las campañas de evaluación y las tareas que se proponen en ellas. También, comentar algunas de las campañas más importantes y describir la campaña en la que se realizó la participación del sistema de detección de depresión.
- Descripción del sistema de detección de depresión para texto en inglés: descripción de las fases que componen el sistema: traducción de los textos, ingeniería de características y fase de experimentación.
- Modificación y adaptación del sistema de detección de depresión y descripción de los cambios realizados.
- Ejecución de los dos sistemas y obtención de resultados. Análisis de los resultados de cada sistema por separado, comentanto las características que más impacto han tenido en cada uno de ellos.

- Comparativa sobre los resultados obtenidos y conclusiones finales para ver el rendimiento de cada sistema.

Capítulo 2

ANTECEDENTES

Para poder situar este proyecto en contexto, primero vamos a realizar un repaso en el campo del PLN, viendo su historia, cómo podemos clasificarlo y sus utilidades. Finalmente veremos cómo podemos relacionarlo con la depresión y ver cómo este puede ser útil en este área.

Además, estudiaremos también qué son las campañas de evaluación y las tareas que se presentan en estas, porque, como se ha dicho, hemos participado con este sistema en una tarea propuesta en una de estas campañas, y es interesante ponerlas en contexto.

2.1. PLN y detección de depresión en medios sociales

2.1.1. Concepto de PLN

El estudio del PLN comenzó en la década de 1950 como un área dentro de la Inteligencia Artificial y la Lingüística, con el objetivo de estudiar los problemas derivados de la generación y comprensión automática del lenguaje natural. Fue en 1950 cuando Alan Turing publicó un artículo que se titula “Computing Machinery and Intelligence” [7] que proponía lo que ahora se conoce como el test de Turing.

El Procesamiento del Lenguaje trata de hacer que las computadoras sean capaces de comprender y generar lenguaje humano, es decir, intenta reducir la brecha comunicacional entre la computadora y el ser humano. Este tiene su origen en la traducción automática cuando en 1954 IBM y la Universidad de Georgetown realizaron el “expe-

rimento de Georgetown", que realizaba la traducción automática de frases del ruso al inglés.

Poco después se realizaron más estudios y experimentos como el programa ELIZA. ELIZA era una simulación del psicoterapeuta Carl Rogers que proporcionaba una interacción muy parecida a la humana. Este programa era capaz de responder a preguntas genéricas de forma solvente. Podría considerarse que fue el primer BOT que existió.

No fue hasta la década de 1990 cuando el enfoque del campo del PLN se trasladó a los métodos basados en el aprendizaje automático y el procesamiento estadístico del lenguaje. Esto fue debido al enorme crecimiento de datos disponibles en línea y el aumento del poder de procesamiento de las computadoras.

En los últimos años, el PLN ha experimentado grandes avances gracias a la aplicación de técnicas de "deep learning" (aprendizaje profundo), como las redes neuronales, que han demostrado un enorme rendimiento en una amplia selección de tareas de PLN.

Habiendo situado históricamente el Procesamiento del Lenguaje Natural, vamos a pasar a ver algunas definiciones de este. Según Elizabeth D. Liddy [8], la definición de Procesamiento del Lenguaje Natural es la siguiente:

> *El Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés) es un conjunto de técnicas computacionales teóricamente motivadas para analizar y representar textos de ocurrencia natural en uno o más niveles de análisis lingüístico con el propósito de lograr un procesamiento del lenguaje similar al humano para una variedad de tareas o aplicaciones.*

Para entender del todo esta definición, tenemos que entender primero que cuando decimos "textos de ocurrencia natural", nos referimos a cualquier tipo de lenguaje, ya sea oral o escrito. El único requisito es que el lenguaje que queramos analizar sea un lenguaje real que utilicen los humanos para comunicarse entre ellos. En segundo lugar, cuando la definición se refiere a "uno o más niveles de análisis lingüístico" es que hoy en día existen muchas formas de analizar un texto.

Otra definición que podemos encontrar del PLN es la siguiente [9]:

> *El procesamiento del lenguaje natural consiste en la habilidad de una máquina para procesar información mediante el uso del lenguaje natural. Se*

puede decir que el PLN consiste en usar una expresión natural que pueda realizar una conversación con una computadora directamente, de forma escrita o de forma hablada.

El PLN se puede dividir en dos categorías: Comprensión del Lenguaje Natural y Generación del Lenguaje Natural (GLN). La Comprensión trata de entender las palabras y las estructuras gramaticales, mientras que la Generación se ocupa de producir texto. La GLN es multidisciplinar y añade conocimientos que provienen de muchas áreas lingüísticas, psicología, ingeniería e informática. El objetivo principal de esta es generar contenidos que intenten alcanzar el nivel del lenguaje natural humano, que sean textos bien estructurados y puedan ser procesados.

El PLN utiliza una combinación de enfoques lingüísticos, estadísticos y computacionales para lograr su objetivo. Esto implica el desarrollo de modelos y algoritmos que pueden aprender a partir de grandes cantidades de datos lingüísticos para comprender las estructuras y reglas del lenguaje. Estos modelos se basan en técnicas de aprendizaje automático, como el aprendizaje supervisado y no supervisado, así como en métodos de procesamiento estadístico del lenguaje.

La ciencia que estudia el lenguaje es la Lingüística, que incluye los campos de la Fonología, que estudia el sonido, la Morfología, que estudia la formación de palabras, la Sintaxis, que estudia la estructura de las oraciones, y la Pragmática, que estudia la comprensión. La lingüística es donde se apoya el PLN para estudiar los textos sean del idioma que sean. Dentro de la Lingüística podemos situar la lingüística computacional (LC), la cual trata de interactuar entre el lenguaje de los humanos y los ordenadores.

En resumen, tanto la primera como la segunda definición vienen a decir lo mismo. El PLN busca que las computadoras sean capaces de tener un procesamiento del lenguaje lo más parecido posible al lenguaje humano. A través de su aplicación, se pretende mejorar la comunicación entre las máquinas y los seres humanos, abriendo la puerta a diversas aplicaciones y desarrollos en el ámbito de la Inteligencia Artificial y la interacción persona-ordenador.

2.1.2. Clasificación del PLN

Para entender cómo funciona el PLN, primero tenemos que entender cómo funciona cada una de sus partes y posteriormente podremos entrar en el funcionamiento de un sistema de PLN.

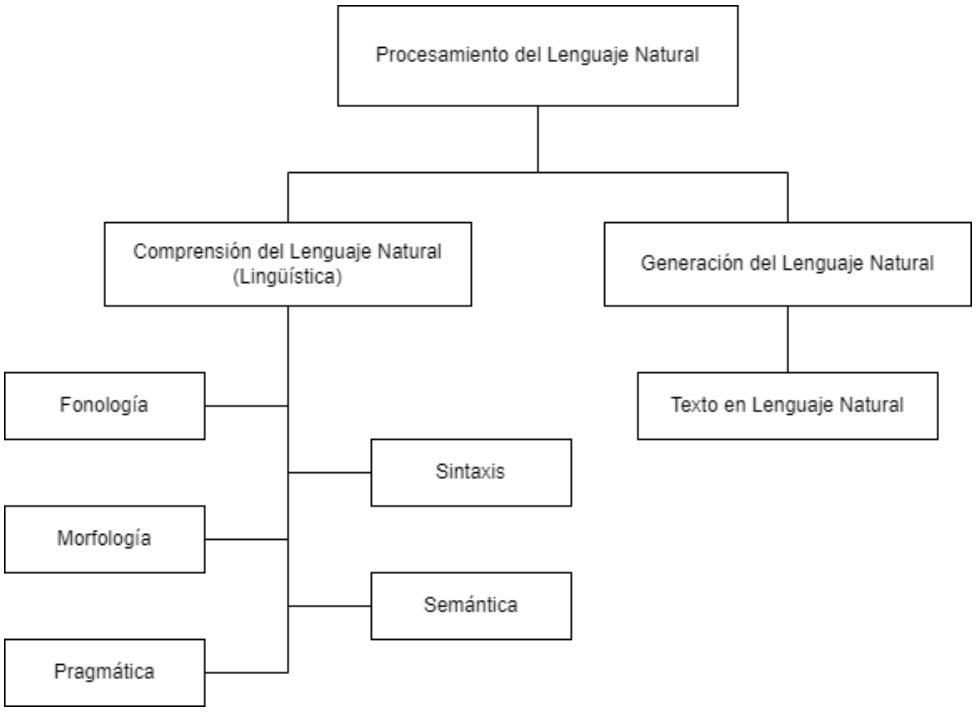

Figura 2.1: Clasificación del PLN.

Como se puede ver en la Figura 2.1 y como he introducido en el apartado anterior, el PLN se puede clasificar en dos partes: La Comprensión del Lenguaje Natural (NLU, por sus siglas en inglés) y la Generación del Lenguaje Natural (NLG, por sus siglas en inglés).

Por tanto, en este apartado, voy a pasar a explicar en qué consiste cada una de las partes y hablar de los elementos principales de cada una de ellas.

Comprensión del Lenguaje Natural (NLU)

La Compresión del Lenguaje Natural extrae conceptos, emociones o palabras clave para poder permitir a las máquinas analizar y comprender el lenguaje natural. Dentro de esta parte se engloba todos los términos que se usan comúnmente en el ámbito del PLN y que forman parte de la Lingüística. Algunos de esos términos son los siguientes:

1. **Fonología**: es la parte de la Lingüística que se refiere a la disposición sistemática del sonido, es decir, al sonido del lenguaje. Esta hace uso semántico del sonido para codificar el significado de cualquier idioma humano.

2. **Morfología**: la morfología comprende la naturaleza de las palabras, los morfemas.

El morfema es la unidad mínima aislable en el análisis morfológico.

Por ejemplo, la palabra mujeres contiene dos morfemas: mujer y -es.

3. **Léxico**: tanto los humanos como los sistemas de PLN interpretamos el significado de las palabras de forma individual. El procesamiento léxico incluye asignar etiquetas de categoría gramatical a las palabras y reemplazar las representaciones semánticas por las propias palabras. Existen técnicas para procesar un texto a nivel léxico como la eliminación de palabras vacías (son aquellas palabras que se repiten con frecuencia en un texto y cuyo significado no es relevante para este), el stemming (es una técnica para reducir las palabras a su raíz) o la lematización (obtener la forma básica de una palabra). Todas estas técnicas ayudan a limpiar el texto y extraer características relevantes de este para poder mejorar la comprensión del significado deseado. Este nivel es fundamental para analizar y comprender el significado de las palabras en PLN.

4. **Sintaxis**: la sintaxis se encarga de analizar la estructura gramatical de una oración. Realizando un análisis sintáctico podemos obtener las dependencias entre las palabras de una oración. El análisis sintáctico analiza el orden de las palabras, la morfología o la categoría gramatical que el análisis léxico no considera.

5. **Semántica**: el objetivo del análisis semántico es determinar el significado correcto de una oración. Para los humanos es muy fácil saber cuál es el significado de una oración gracias a todo el conocimiento de la lengua que hablamos, pero, al contrario que nosotros, las máquinas no tienen esa facilidad. Necesitan de un proceso de análisis semántico para determinar el significado correcto de la oración de varios significados posibles que pueda tener. Para hacerlo intentan buscar la relación que tienen las palabras más importantes de la oración y procesar la estructura lógica de la oración.

6. **Discurso**: el discurso se encarga, al igual que la sintaxis, del análisis de la estructura lógica, pero en este caso no solo de una oración, sino de un texto más amplio. Busca el significado más adecuado para un texto buscando las relaciones entre las diferentes oraciones que lo componen.

7. **Pragmática**: el objetivo de la pragmática es entender el conocimiento que viene de fuera del texto, es decir, se centra en el contexto de la oración y cuando a una oración diferentes personas pueden darle significados diferentes, aparece la ambigüedad. La ambigüedad es el mayor problema del lenguaje natural y como he dicho significa que a una oración se le pueden dar diferentes significados o interpretaciones. Esta puede aparecer de diferentes formas: analizándola sintácticamente, léxicamente o semánticamente.

Estos son los términos más importantes en referencia a la Comprensión del Lenguaje Natural y es muy importante entenderlos porque conforman una de las partes más importantes de un sistema de PLN, que es el análisis de los datos y el preprocesamiento de los mismos.

Generación del Lenguaje Natural (NLG)

La segunda parte del PLN es la Generación del Lenguaje Natural (NLG). Esta parte consiste en el proceso de generar frases, oraciones o textos que tengan un significado a partir de una representación interna.

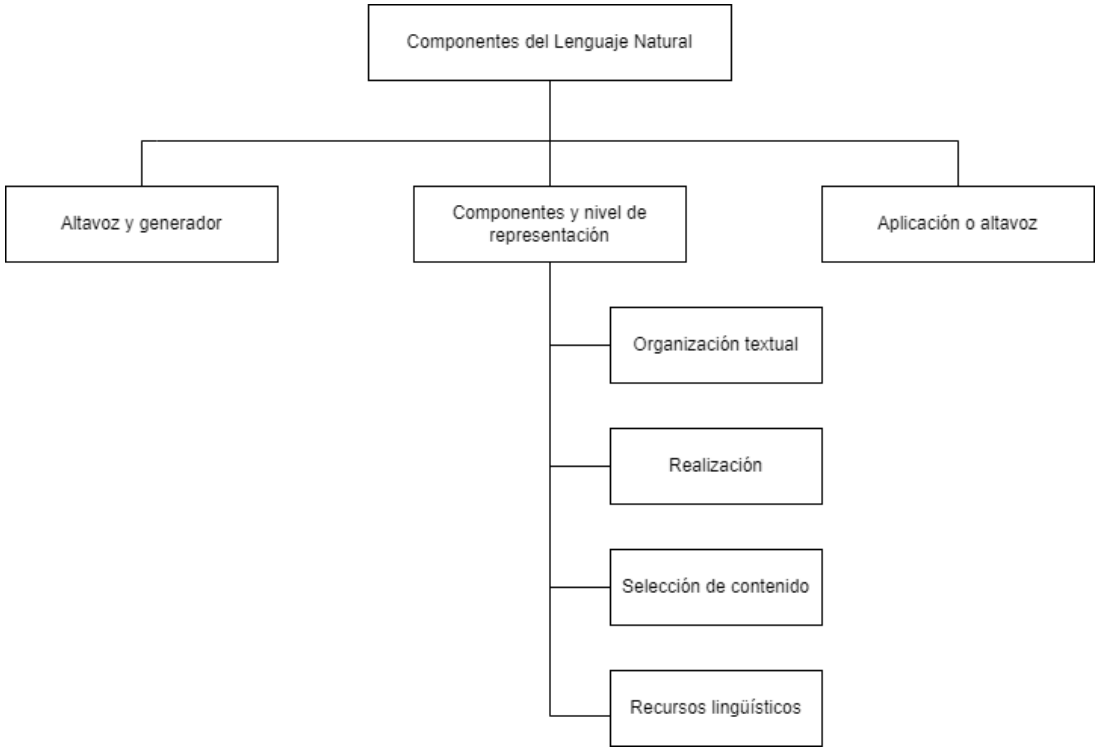

Figura 2.2: Componentes del NLG.

Para poder producir texto, tal y como se puede observar en la Figura 2.2, necesitamos un generador que sea capaz de producir la frase que queremos adecuada al contexto en el que nos encontremos. Después, necesitamos una aplicación o un altavoz (dependiendo de la forma en que queramos producir el texto) que nos permita ver o escuchar el texto que el generador ha producido.

Una vez sabido esto, para generar lenguaje natural necesitamos realizar algunas tareas:

1. **Selección de contenidos**: una vez hemos realizado un análisis sintáctico del

texto, tenemos que seleccionar el contenido más relevante para nosotros y eliminar el contenido que no nos interesa.

2. **Organización textual**: en términos de la lingüística, la información se debe presentar correctamente de acuerdo a la gramática.

3. **Recursos lingüísticos**: si queremos ser más completos, podemos apoyar nuestro sistema de generación de textos con recursos lingüísticos relacionados con la tarea que estamos realizando.

4. **Realización**: una vez hemos seleccionado los contenidos y los hemos organizado, habiendo añadido también los recursos lingüísticos que hemos considerado de ayuda, necesitamos mostrar el texto de forma real por medio de alguna salida en una pantalla o por medio de un altavoz en forma de sonido.

2.1.3. Aplicaciones del PLN

Una vez hemos entendido las partes en las que se clasifica el PLN, vamos a pasar a comentar algunas de las muchas áreas en las que se puede aplicar. En este apartado, vamos a examinar cómo el PLN se utiliza para mejorar la precisión y eficacia de los motores de búsqueda o cómo ha facilitado la traducción automática y la comprensión de idiomas.

1. **Traducción automática**: traducir un texto no se basa sólo en traducir las palabras de forma individual, si no que tenemos que mantener un significado para toda la oración manteniendo la gramática y los tiempos verbales de forma correcta. Este es el mayor desafío de las tecnologías de traducción automática actuales. Las traducciones se realizan con motores estadísticos haciendo uso por ejemplo del aprendizaje profundo. Estos motores recopilan la mayor cantidad de datos que puedan de los dos idiomas que intervienen en la traducción y los procesa para determinar la probabilidad de que algo en el idioma 1 se corresponda con algo en el idioma 2. Un ejemplo de esto es Google, que en 2016 anunció un nuevo sistema basado en redes neuronales y aprendizaje automático.

2. **Categorización de textos**: esta es la aplicación en la que se basa este Trabajo Fin de Grado. La categorización de textos trata de asignar categorías predefinidas a los textos que queramos categorizar. Un buen ejemplo puede ser la detección de spam en nuestro correo electrónico. En el caso de este proyecto, vamos a hablar sobre un sistema de categorización de depresión en textos (escritos en

español), de forma que al sistema se le pasa una entrada (un conjunto de textos escritos por un usuario cualquiera) y el sistema determinará si el usuario que lo ha escrito tiene indicios de depresión o no.

3. **Extracción de información**: esta aplicación consiste en extraer datos o información específica a partir de un conjunto de textos. Esta información se extrae para mostrarla de forma estructurada y se extrae de documentos no estructurados o semi-estructurados, es decir, se transforma esa información no estructurada en una base de datos estructurada. Esta aplicación se usa por ejemplo para realizar análisis de opiniones de clientes: extraes la información del sitio web, por ejemplo, de un archivo HTML (documento semi-estructurado) y la transformas en una base de datos estructurada para poder aplicar técnicas de análisis de datos de tus clientes.

4. **Recuperación de información**: la recuperación de información consiste en que, a partir de un gran conjunto de documentos, extraemos a través de una consulta un subconjunto del conjunto inicial que contiene la información más relevante para la consulta. El mejor ejemplo de un sistema de recuperación de información son los buscadores web como Google.

5. **Resumen de textos**: consiste en sintetizar la información relevante y más importante de un texto, de manera que podamos reducir su longitud manteniendo su significado esencial. La finalidad de esta aplicación es proporcionar una versión concisa y comprensible del texto original, para permitir al lector una comprensión rápida de la información.

Estas son algunas de las aplicaciones más importantes del PLN. Como podemos ver muchas de ellas están presentes en nuestro día a día, como la recuperación de información en los motores de búsqueda.

2.1.4. Cómo aplicar el PLN para detectar la depresión

Habiendo comentado algunas de las muchas aplicaciones del PLN, vamos a ver cómo se puede aplicar a la detección de depresión, que es el tema que vamos a abordar.

Primero tenemos que saber cómo se detecta la depresión. En una situación normal, el proceso para diagnosticar la depresión es el siguiente:

1. El psicólogo o psiquiatra identifica los síntomas en una consulta con el paciente. Estos síntomas pueden variar según el tipo de depresión de la persona. Pueden ser síntomas más relacionados con la mente, como la falta o el exceso de sueño, cambios bruscos de actitud; o síntomas más físicos, como marcas de autolesiones.

2. El profesional que ha identificado los síntomas realiza las pruebas pertinentes, analiza los resultados y diagnostica el tipo de depresión que padece el paciente.

Pues bien, en estas situaciones, el uso de herramientas adecuadas para la evaluación del diagnóstico final puede ayudar de forma significativa al diagnóstico final. En esta parte es donde podemos utilizar los medios sociales para ayudar a detectar la depresión.

El problema es cuando los propios pacientes se niegan a ir al hospital o a clínicas privadas por ellos mismos porque no quieren aceptar que tienen depresión. Debido a esto, los psicólogos y psiquiatras se están esforzando en identificar la depresión a través de aplicaciones para móviles. Gracias al crecimiento tan grande de las redes sociales (como Facebook o Twitter), hay una enorme cantidad de publicaciones que además se generan en tiempo real. Esto es lo que ha llevado a intentar analizar la salud mental de la gente basándonos en datos de texto.

Todo este texto nos da la posibilidad de analizar las emociones humanas y los pensamientos de las personas pudiendo así detectar indicios de depresión a través del PLN. De esta forma podemos, por ejemplo, separar los textos en frases y buscar similitudes palabra a palabra o podemos clasificar las frases en función del tipo de problema asociado a la depresión que tenga esa frase. Incluso se han realizado estudios para crear diccionarios con vocabulario relacionado con la depresión para detectar estos síntomas. Por ejemplo, el estudio realizado por Hyesil Jung, Hyeoun-Ae Park y Tae-Ming Song [10], donde se desarrolló una ontología sobre depresión en adolescentes con una terminología que comprendía 1682 sinónimos de 443 clases.

Para clasificar un texto necesitamos un modelo de clasificación previamente entrenado con un conjunto de textos que ya están categorizados. Por ejemplo, en un estudio realizado por Jong Woo Kim en la Universidad Hanyang de Seúl [11], Corea del Sur, proponen un sistema de clasificación que se divide en tres pasos:

1. El primer paso se llama “fase de entrenamiento de clasificación de oraciones”. En esta fase se va a entrenar un modelo usando BERT, Word2Vec y word embedding. Se recoge un conjunto de textos de Internet, se separan en oraciones

que serán preprocesadas eliminando palabras vacías, etc. Por último, se leen las frases de manera individual determinando si estas están relacionadas o no con la depresión. En caso afirmativo, se le dará un valor del 0 al 9 en función de que síntoma/s se refleja/n en esa oración. Esto se hace usando el Patient Health Questionnaire-9 (PHQ-9).

2. El segundo paso se llama “fase de entrenamiento del clasificador de depresión”. En esta fase se realiza lo mismo que en la primera, pero en esta se usaron textos de redes sociales. Estos textos se extrajeron de 30 adultos diferentes que tenían depresión (basándose en el PHQ-9) y de 30 adultos que no tenían depresión. Se preprocesaron los textos igual que en la primera fase y se clasificaron de igual forma que en la primera fase. Finalmente, un algoritmo de regresión logística se entrenó con estos datos.

3. Por último, la tercera fase que se llama “fase de clasificación de depresión de los usuarios”. En esta fase se usaron los textos extraídos durante dos semanas de los usuarios de las redes sociales, después se preprocesaron los textos de igual forma que en la primera fase y se clasificaron usando los modelos pre entrenados. Con los resultados de las clasificaciones se calcularon los valores de las variables de entrada del clasificador de regresión logística. Haciendo uso de estas variables, se categorizaron a los usuarios con depresión o no.

Una vez se obtuvieron todos los resultados se hizo el análisis de los resultados comparando los resultados obtenidos con los resultados reales para ver el desempeño del modelo. Este modelo que usaron era un modelo más complejo que el modelo del que trata este proyecto.

El modelo del que trata este Trabajo Fin de Grado es un modelo que se ha usado para participar en una campaña de evaluación, concretamente en la tarea 2 de MentalRiskES que se llama “Depression detection”. De esto se va a hablar más extendido en el siguiente capítulo.

Este modelo está realizado de forma que solo acepta como textos de entrada textos escritos en inglés. El modelo se ha entrenado con diferentes algoritmos para ver cual daba mejores resultados y es un modelo de clasificación binaria, es decir, al contrario que con el modelo que he explicado anteriormente, este no asigna una puntuación en función de los síntomas que se puedan encontrar en el texto. Simplemente realiza una clasificación se “Sí” o “No” en función de si el usuario que ha escrito el texto tiene indicios de depresión o no.

Además, este modelo no funciona de forma que trata los textos de forma individual.

El modelo recoge un conjunto de posts de un usuario concreto y trata todos los posts de manera conjunta, es decir, no clasifica un texto individual si no que clasifica al usuario con el total de textos que ha escrito. Más adelante se desarrollará esta explicación de forma más extendida.

2.2. Campañas de evaluación y sus tareas

Antes de hablar del sistema de clasificación que se ha usado para este trabajo, primero voy a comentar qué son las campañas de evaluación y en cuál de ellas se ha realizado la participación con este sistema.

2.2.1. ¿Qúe son las campañas de evaluación?

Las campañas de evaluación son iniciativas que se llevan a cabo para evaluar y comparar el rendimiento de diferentes modelos, sistemas o algoritmos en tareas específicas relacionadas con el Procesamiento de Lenguaje Natural. Estas campañas son organizadas y sistemáticas, y su objetivo principal es medir y demostrar el progreso en la tecnología del PLN, así como promover la colaboración y la competencia en la comunidad de investigación.

Estas campañas son fundamentales para el avance de la tecnología en PLN y para proporcionar una evaluación comparativa de sistemas en una amplia variedad de tareas relacionadas con el lenguaje natural, lo que contribuye al desarrollo continuo de soluciones de PLN más efectivas y precisas.

Las campañas de evaluación se componen de unas características clave que es importante conocer:

1. **Definición de las tareas específicas**: una campaña de evaluación se centra en unas tareas concretas de PLN. Estas tareas pueden abarcar una amplia gama de aplicaciones, desde traducción automática, resumen de texto, análisis de sentimientos, hasta reconocimiento de voz y respuesta a preguntas, entre otras. Cada tarea se selecciona cuidadosamente para reflejar desafíos reales en el Procesamiento del Lenguaje Natural.
2. **Conjunto de datos de referencia**: para poder evaluar el rendimiento de los sistemas de PLN, se crean conjuntos de datos de referencia, que contienen ejem-

plos de textos etiquetados o anotados manualmente. Estos conjuntos de datos sirven como punto de referencia para comparar el rendimiento de diferentes sistemas. Los ejemplos pueden variar en tamaño y complejidad, y a menudo se utilizan tanto para entrenar como para evaluar los sistemas.

3. **Métricas de evaluación**: las campañas de evaluación definen unas métricas específicas para medir el rendimiento de los sistemas en las tareas de PLN. Estas métricas son esenciales para cuantificar la calidad de las respuestas o soluciones proporcionadas por los sistemas. Las métricas pueden variar según la tarea, pero comúnmente incluyen precisión, recuperación, F1-score, BLEU score, ROUGE score y otras métricas específicas para cada tarea. Muchas de estas métricas las explicaré más tarde cuando hable del análisis de resultados del modelo de clasificación.

4. **Participantes**: las campañas de evaluación atraen la participación de investigadores, desarrolladores y equipos de todo el mundo. Estos participantes compiten entre sí para desarrollar sistemas que superen a los sistemas existentes o para abordar los desafíos planteados en las tareas específicas de PLN.

5. **Competencia y colaboración**: aunque la competición es un aspecto importante de estas campañas, también fomentan la colaboración. Los participantes a menudo comparten sus enfoques, técnicas y lecciones aprendidas, lo que contribuye al avance colectivo de la investigación de PLN.

6. **Evaluación objetiva**: la evaluación de rendimiento se basa en datos objetivos y resultados medibles. Esto permite identificar cuáles son los enfoques más efectivos y cuáles tienen margen de mejora, lo que impulsa la innovación y el perfeccionamiento de las tecnologías de PLN.

2.2.2. Ejemplos de campañas de evaluación

Existen muchas campañas de evaluación conocidas tanto a nivel mundial como a nivel nacional. Primero voy a comentar algunas campañas a nivel internacional y después me voy a centrar en las campañas de evaluación en España y finalmente me centraré en la campaña de evaluación y tarea en la que se ha participado con el modelo de clasificación.

Algunas de las campañas de evaluación más conocidas a nivel mundial son las siguientes:

1. **Conferencia de Recuperación de Información y Evaluación de Texto (TREC)**[12]: esta es una de las campañas más antiguas y conocidas en el ámbito de PLN a nivel internacional. TREC se enfoca en la evaluación de sistemas de recuperación de información y ha sido crucial para avanzar en la investigación en este campo.

2. **Evaluación de Traducción Automática (WMT)**[13]: WMT se centra en la evaluación de traducción automática y se lleva a cabo a nivel internacional. Los participantes compiten para mejorar la calidad de las traducciones automáticas en varios idiomas.

3. **Conferencia de Análisis de Sentimiento y Opinión (Semeval)**[14]: Semeval se enfoca en el estado del arte del análisis semántico y contribuye a la creación de conjuntos de datos anotados de alta calidad en una serie de problemas cada vez más difíciles de la semántica del lenguaje natural.

4. **Diálogo Orientado a Objetivos (DSCT-Dialogue State Tracking Challenge)**[15]: este es un ejemplo de una campaña de evaluación centrada en sistemas de diálogo orientado a objetivos, donde se evalúan sistemas de Procesamiento del Lenguaje Natural que pueden llevar a cabo conversaciones con un propósito específico.

5. **Evaluación de Tecnologías del Lenguaje Italiano (Evalita)**[16]: aunque no es una campaña de evaluación a nivel mundial, Evalita es un ejemplo de una campaña de evaluación a nivel europeo que se enfoca en tecnologías del lenguaje natural en italianio.

Para terminar con las campañas de evaluación a nivel internacional, voy a hablar de una de las campañas de evaluación internacionales más destacada con más detalle, el **Conference and Labs of the Evaluation Forum (CLEF)** [17]:

1. **Conference and Labs of the Evaluation Forum (CLEF)**: esta campaña de evaluación es una de las iniciativas más destacadas a nivel internacional para promover la investigación y el desarrollo de sistemas de PLN en un contexto multilingüe. El foro CLEF se originó en Europa y ha influido en gran medida en el avance de las tecnologías del lenguaje natural en toda la comunidad de investigadores y desarrolladores de PLN. Esta campaña de evaluación de inició en el año 2000 bajo el nombre de Cross-Language Evaluation Forum, como un evento en conjunto con European Conference for Digital Libraries (ECDL), que ahora se llama Theory and Practice on Digital Libraries (TPDL) [18]. Desde 2010, CLEF

se desarrolló como evento independiente pasando a llamarse Conference and Labs of the Evaluation Forum

Dentro de CLEF tenemos el workshop Early risk prediction on the Internet (eRisk) que se centra en la exploración de la metodología de evaluación, los indicadores de efectividad y las aplicaciones prácticas, especialmente aquellas relacionadas con la salud y la seguridad, de la detección temprana de riesgos en Internet.

Desde sus primeras ediciones eRisk se enfocó en la identificación de signos tempranos de problemas de salud mental, como la depresión, el suicidio y otros trastornos relacionados. A lo largo de los años, las tareas propuestas en eRisk se han vuelto más complejas. En 2018 las tareas propuestas eran dos: la "detección temprana de signos de depresión" y la "detección temprana de signos de anorexia", mientras que cada año aparecían más tareas y estas eran más específicas. Por ejemplo, en 2019 apareció una tarea en la que se medía el nivel de severidad de los signos de depresión, es decir, ya no solo se medía la detección temprana si no que, a su vez, se pedía la gravedad de los signos detectados.

También, este workshop se ha ido adaptando a las nuevas tendencias. Un ejemplo es la tarea que se propuso en 2021 donde se medía la detección temprana de signos de ludopatía. En 2022, se incluyó una tarea en la que se pedía medir la gravedad de los signos detectados de desórdenes alimenticios.

Una vez comentados las campañas de evaluación a nivel internacional, pasamos a comentar algunas de las campañas de evaluación que existen a nivel nacional. En España tenemos algunas que también son importantes, como son las siguientes:

1. **Taller de Análisis Semántico en la SEPLN (TASS)**[19]: este taller anual organizado por la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN) se enfoca en la evaluación de sistemas de análisis de sentimiento y opinión en español.

2. **IberEval**[20]: Esta campaña tiene como objetivo fomentar y promover el desarrollo de las Tecnologías del Lenguaje Humano (HLT) para las lenguas ibéricas (español, portugués, catalán, euskera y gallego), mediante la creación de series de evaluación y de un foro de discusión sobre sistemas de Procesamiento del Lenguaje Natural de forma continuada.

Y, por último, voy a hablar de **Iberian Languages Evaluation Forum (IberLEF)** que es la campaña donde se ha participado usando el modelo de clasificación de depresión

del que trata este proyecto. Esta camapaña surgió a raíz de las dos mencionadas anteriormente. A partir del año 2019 se unieron para crear IberLEF

Iberian Languages Evaluation Forum (IberLEF) [21] es una campaña de evaluación que se celebra cada año en el congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN) desde 2019. Es una campaña de evaluación en la que se realizan tareas no solo en español, sino que también se realizan tareas en otras lenguas ibéricas (Portugués, Catalán, Vasco y Gallego). Esta campaña busca fomentar la investigación en labores relacionadas con el procesamiento, la comprensión y la creación de textos.

En IberLEF, la comunidad de investigadores establece nuevos retos de investigación y propone tareas para avanzar en el estado del arte del PLN. Los organizadores de las tareas aceptadas diseñan el proceso de evaluación de acuerdo a la propuesta presentada, impulsan las tareas y gestionan el envío y la revisión científica de los trabajos y sistemas presentados por los investigadores o equipos que participan en cada tarea. Estas tareas se revisan inicialmente por miembros de los comités de dirección y programa de IberLEF. Posteriormente, estas tareas son evaluadas por los presidentes generales de IberLEF. Por último, después de la revisión de los sistemas presentados en cada tarea, los organizadores informan de los resultados a los participantes y estos deben presentar una descripción de su sistema en el workshop de IberLEF.

Aunque esta campaña de evaluación era nacional, actualmente puede considerarse como una campaña de evaluación internacional. Para poner esto en contexto, vamos a ver el siguiente gráfico con los datos de participantes por país del congreso realizado en septiembre de este año 2023.

Como podemos observar en el gráfico de la Figura 2.3, aunque la mayor parte de participantes son de habla hispana (72 grupos españoles, 64 grupos mejicanos y 19 chilenos), han participado grupos de investigación de diversos países. En total, han participado 211 grupos de investigación en el congreso de este año de todos los continentes.

Además, los organizadores de las tareas tienen que presentar un resumen de su ejercicio de evaluación de la tarea, que posteriormente son publicados en la revista de la SEPLN después de una revisión de estos realizada por el comité organizador de IberLEF.

En cuanto a las tareas que se presentan, hay de varios tipos. Por ejemplo, para el congreso realizado el 26 de septiembre de 2023 en Jaén, se han presentado tareas que se han dividido en siete apartados. Estos han sido:

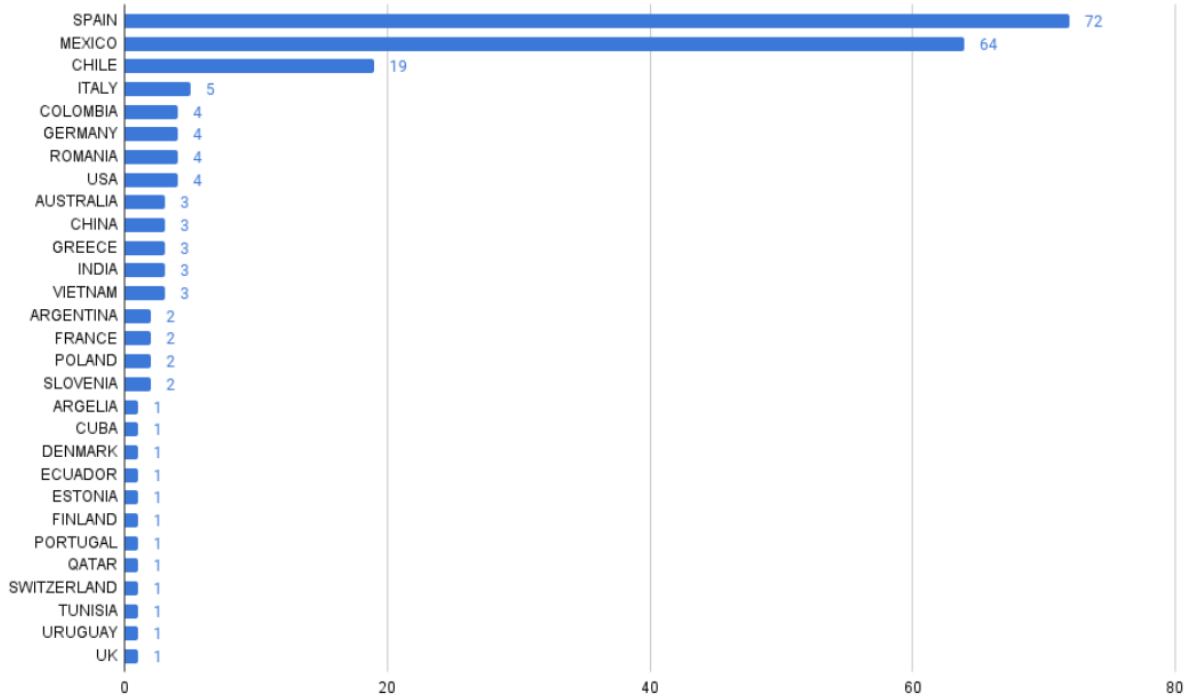

Figura 2.3: Número de participantes por país en IberLEF 2023.

1. **Identificación de textos generados automáticamente**.

2. **Contenido clínico**: contenido relacionado con el mundo de la medicina.

3. **Análisis de cambio de código**.

4. **Predicción temprana del riesgo en Internet**: en este apartado es donde se encuentran las tareas de MentalRiskES, que voy a comentar posteriormente y donde se encuentra la tarea donde hemos participado con el sistema de clasificación.

5. **Contenidos nocivos e inclusivos**.

6. **Ideología política y propaganda**.

7. **Sentimientos, posturas y opiniones**.

Como he indicado, el apartado que es importante para este proyecto, es el apartado de Predicción temprana del riesgo en Internet, donde se encuentran las tareas de MentalRiskES: Early detection of mental disorders risk in Spanish, es decir, Predicción temprana de riesgo de desórdenes mentales en español. Esta tarea se divide en 3 tareas: Detección de desórdenes alimenticios, Detección de depresión y Detección de desórdenes desconocidos.

Nosotros nos vamos a centrar en la segunda tarea de las tres: Detección de depresión. Esta subtarea, a su vez, también se divide en varias subtareas, concretamente cuatro. Yo voy a describir y comentar la subtarea en la que hemos participado, que es la tarea de Clasificación binaria. Como su nombre indica, en esta tarea el objetivo es identificar si el usuario tiene depresión o no. En el conjunto de datos proporcionado por la organización encontraremos una columna que indicará si el usuario sufre depresión o no a través de los valores 0 para “control” (el usuario no sufre depresión) y 1 para “sufre” (el usuario sufre depresión).

Una vez descritas algunas de las campañas de evaluación más importantes a nivel internacional y nacional, y habiendo introducido la tarea objetivo de este proyecto y del sistema de clasificación que se ha usado para esta, en el siguiente capítulo pasaremos a ver el sistema de clasificación en detalle y posteriormente se comentarán todos los cambios realizados para que el sistema funcione sin necesidad de realizar una traducción previa de los textos y funcione con textos en español.

Capítulo 3

MATERIALES Y MÉTODOS

En este capítulo, vamos a estudiar el conjunto de datos proporcionado por la campaña de evaluación IberLEF 2023 [21] para la tarea de detección de depresión (clasificación binaria) de MentalRiskES [22]. También se analizará en detalle el funcionamiento del sistema del que trata este proyecto y, posteriormente, se realizarán todos los cambios pertinentes para que este sistema funcione con un conjunto de texto de entrada en español.

3.1. Sistema de clasificación para la detección de depresión

Antes de empezar a hablar del sistema y cómo funciona, primero hay que decir que este sistema ha sido desarrollado por el grupo TextualTherapists para la participación en la campaña de evaluación IberLEF, en la tarea de Detección de Depresión de MentalRiskES. Este grupo está formado por 6 integrantes, en los cuales me incluyo. Los integrantes son: Alberto Fernández Hernández, José Viosca Ros, Raquel Enrique Guillén, Noa P. Cruz Díaz, Salud María Jiménez Zafra (tutora de este TFG) y, por último yo mismo, Raúl Moreno Sánchez.

El funcionamiento de estos sistemas a simple vista es un funcionamiento sencillo. Como entrada al sistema se le pasan una serie de textos, el sistema los analiza y como salida nos dice si la persona que ha escrito los textos tiene indicios de depresión o no. En este caso, lo que queremos no solo es que detecte la depresión, si no que la detecte de manera temprana, es decir, si un usuario ha escrito 10 textos, el sistema es mejor si es capaz de saber si el usuario tiene depresión cuando lee el cuarto texto, que

cuando lee el sexto (obviamente sabiendo que el usuario tiene depresión, si detecta que tiene depresión, pero realmente no la tiene, no es efectivo).

Una parte importante de un sistema de clasificación es el entrenamiento de este. Un buen conjunto de datos es muy importante en un sistema de clasificación. Aunque el código añada muchas funcionalidades, un mal conjunto de datos puede hacer que nuestro sistema sea deficiente. En nuestro caso, el conjunto de datos está proporcionado por la campaña de evaluación y es el mismo para todos los participantes. De esta forma, teniendo el mismo conjunto de datos para entrenar nuestro sistema, la diferencia de si es mejor o peor que los demás vendrá dada por el diseño de este.

Estos conjuntos de datos son proporcionados por la organización para probar nuestros sistemas y mejorarlos. Tenemos un tiempo determinado para realizar esto y, una vez llegada una fecha concreta, la organización proporciona conjuntos de datos por rondas. Estos conjuntos de datos son los que tenemos que clasificar y enviar a la organización. Se proporcionan por rondas porque, como ya he dicho antes, esta tarea se basa en la detección temprana de la depresión, por lo que, si tu sistema es capaz de clasificar al usuario durante las primeras rondas (con menos mensajes), el sistema será mejor que el de los demás participantes.

Vamos a dividir en varios apartados el funcionamiento del sistema, siguiendo el flujo de ejecución que hemos seguido de cada archivo. Como sabemos, los datos se proporcionaban por rondas, y el objetivo, es detectar de forma temprana la depresión. Esto se ha hecho de la siguiente forma:

1. Primero se comprueba cuántas rondas tenemos. En nuestro caso cada ronda, equivalía a un mes del año y, en total, eran 12 rondas. Lo primero que hacemos es ver si hemos llegado al total de rondas y agrupar todos aquellos mensajes pertenecientes a usuarios que no han sido clasificados como “sufren de depresión”.

2. Una vez agrupados los mensajes anteriores, realizamos la traducción e ingeniería de características de estos. Esta traducción la hacemos porque el sistema de detección de depresión temprana del que partimos solo funciona con texto de entrada en inglés. Por lo que es necesaria una traducción previa de los textos a inglés para poder realizar las predicciones.

3. Después, agrupamos los textos por usuarios, ya que, hasta aquí, hemos trabajado con textos individuales, pero nuestro objetivo es clasificar al usuario no al texto individual.

4. Finalmente, entrenamos los modelos que queremos y realizamos la predicción. En nuestro caso, para esta fase del flujo de ejecución, hemos usado la biblioteca PyCaret. Esta biblioteca para Python nos permite elegir entre un conjunto de modelos para entrenar y quedarnos con los mejores modelos en función de la variable que queremos maximizar. Para terminar, realizamos la predicción y subimos los resultados, además de las métricas de rendimiento pedidas por la organización. Esto se ha hecho, para ver el consumo de los modelos usando una biblioteca de Python que detecta el hardware del ordenador donde se ha entrenado y ejecutado el modelo y calcula las emisiones producidas. De esta forma, si por ejemplo tenemos un modelo muy bueno, pero tiene un consumo y emisiones muy altas, no es un modelo eficiente.

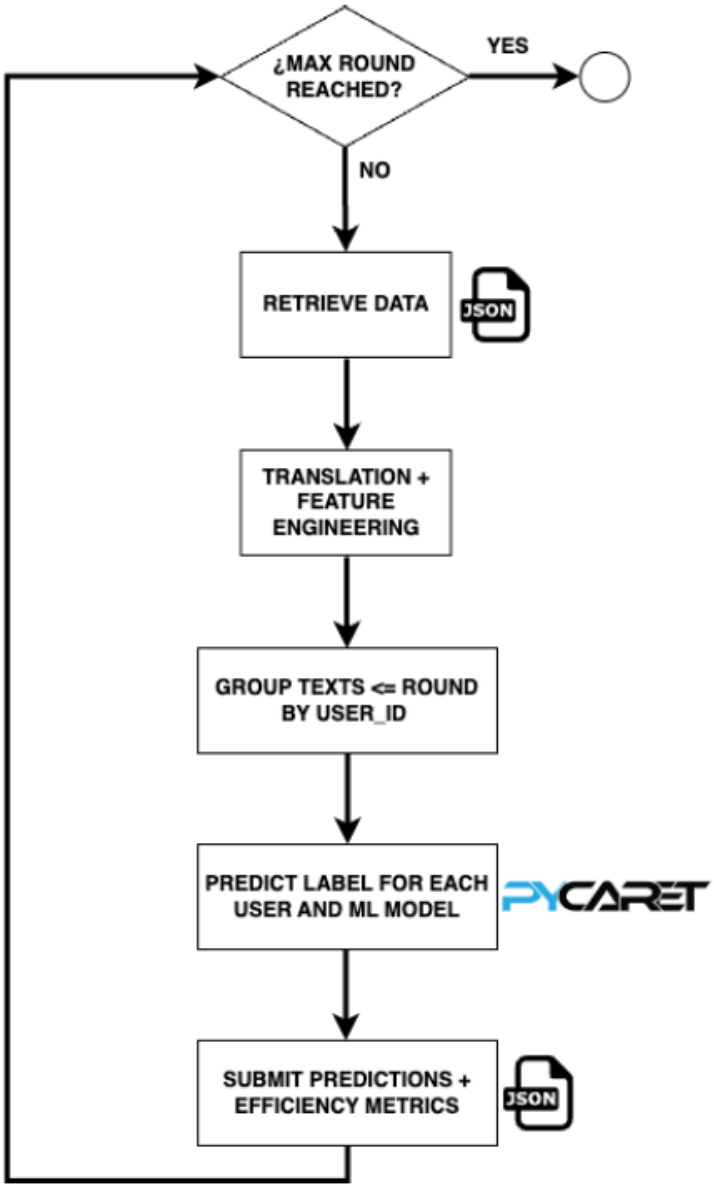

Figura 3.1: Diagrama de flujo de ejecución.

Como se aprecia en la Figura 3.1, podemos ver paso por paso el flujo explicado anteriormente de manera más intuitiva.

Sabiendo lo anterior, vamos a analizar el conjunto de datos, que se divide en tres partes, el conjunto de prueba, el de entrenamiento y el de test (el de test es el único que no está etiquetado).

3.1.1. Análisis del conjunto de datos

El conjunto de datos está formado por mensajes de Telegram en español. Se han recogido mensajes de 334 conversaciones de usuarios de grupos públicos de Telegram. Todos los mensajes son anónimos y están etiquetados de forma que “0” indica que el mensaje está clasificado como “control” (esto indica que no sufre depresión) o “1” que indica “sufre” (si sufre depresión). Como cada usuario tiene un número de mensajes, la forma de saber si el usuario sufre depresión se calcula de la siguiente forma:

- Vamos a llamar al número total de mensajes etiquetados a 1 de un usuario “D” y al número total de mensajes de un usuario “T”. Sabiendo esto, la probabilidad de que una persona sufra depresión, “P”, se calcula como el número total de mensajes etiquetados a 1 entre el número total de mensajes del usuario, es decir:

$$P = \frac{D}{T} \tag{3.1}$$

  Si una persona tiene todos sus mensajes etiquetados a 0, el resultado será 0, que significa un 100 % de no sufrir de depresión. Si, por el contrario, tiene todos sus mensajes etiquetados a 1, el resultado será 1, que significa un 100 % de sufrir depresión.

La distribución de los mensajes del conjunto de datos se muestra en la siguiente tabla:

	trial	**train**	**test**	**total**
sufre	6	94	-	100
control	4	81	-	89
total	10	175	149	334

Tabla. 3.1: Distribución de los mensajes en el conjunto de datos

Un detalle importante de la Tabla 3.1 es que los números no indican el número de mensajes del conjunto de datos, si no el número de usuarios que han escrito mensajes del conjunto de datos.

A continuación, vamos a ver un Análisis Exploratorio de los Datos (EDA), para ver más en detalle las características del conjunto de datos.

Los ficheros de prueba y entrenamiento (trial y train) tienen la misma estructura. Están formados por cuatro columnas que contienen los siguientes datos:

- **id_message**: el identificador único del mensaje.
- **message**: el texto del mensaje.
- **date**: la fecha de publicación del mensaje.
- **user_id**: el identificador único del usuario que ha escrito el mensaje.

En la siguiente tabla podemos ver algunas características de los dos conjuntos de datos:

Fichero	Máximo caracteres	Mínimo caracteres	Mensajes con más de 100 caracteres	Mensajes con menos de 100 caracteres	Número total de mensajes
trial	1024	14	87	537	624
train	4279	7	1000	5248	6248

Tabla. 3.2: Datos sobre los caracteres de los mensajes del conjunto de datos

De la Tabla 3.2, podemos extraer algunas conclusiones. La mayor parte de los mensajes de cada conjunto de datos son mensajes con menos de 100 caracteres, es decir, son mensajes cortos. Esto es normal, ya que son mensajes de chats de Telegram, no son posts en ninguna otra red social. Pero esto, no es indicativo de que tengan poca información que podamos extraer para predecir si el usuario tiene depresión o no. Al final, donde un usuario puede mostrar más sentimientos, es en este tipo de redes sociales, como Telegram o Whatsapp. Redes sociales como Twitter pueden ser más ambigüas, porque una persona puede mostrar sentimientos en esas redes que no son los que realmente siente.

En cuanto a los mensajes por usuario, he calculado cuántos usuarios tienen más de 35 mensajes y menos de 35 mensajes recorriendo los mensajes de cada fichero

de datos, tanto el de trial como el de train. He elegido esta cifra teniendo en cuenta el número máximo de mensajes que tiene un usuario para ver la proporción de mensajes que ha mandado cada usuario. De esta forma, podemos saber si la cantidad de mensajes de cada usuario es una buena cantidad para el entrenamiento del modelo. A partir de esto, he obtenido la siguiente tabla:

De la Tabla 3.3, obtenemos que alrededor de un 70 % de los usuarios tienen menos de 35 mensajes. Esto nos puede indicar que no tenemos demasiados mensajes para poder entrenar nuestro modelo, pero como el objetivo es clasificar al usuario y no un mensaje, al agrupar los mensajes por usuario es más efectivo detectar la depresión, porque podemos encontrar comportamientos relacionados con la depresión en un conjunto mayor de mensajes.

Fichero	**Usuarios con más de 35 mensajes**	**Usuarios con menos de 35 mensajes**	**Número total de usuarios**
trial	6	4	10
train	57	118	175
Total	63	122	185

Tabla. 3.3: Número de mensajes por usuario

Otro dato importante son los emoticonos. Hoy en día se usan mucho los emoticonos en aplicaciones de mensajería como Whatsapp o Telegram. Los mensajes del conjunto de datos tienen emoticonos, pero en formato texto, es decir, si un mensaje tiene un emoticono de una cara triste, no aparece el emoticono como tal, sino el texto “cara triste”. Esto nos da también la posibilidad de analizar estos emoticonos con alguna herramienta añadida al sistema para analizar de una forma más efectiva al usuario.

Habiendo explicado todo esto, nosotros modificamos los conjuntos de datos y añadimos algunas columnas para analizar mejor los mensajes de texto. Creamos dos ficheros añadiendo las siguientes columnas extra:

- **label**: esta columna representa el valor “0” o “1” que indica si ese mensaje es de un usuario que sufre o no depresión. Esta columna se añade para poder realizar el entrenamiento sin tener que usar el fichero gold que proporciona la organización, donde aparece cada “user_id” asociado a su etiqueta.
- **num_uppercase_words**: esta columna indica el número de palabras que comienzan con mayúscula del mensaje.

- **message_emojized**: contiene el mismo texto que la columna "message" pero sustituyendo las palabras que indican un emoticono por el emoticono en sí.
- **fe_num_pos_emojis**: indica el número de emoticonos con valor sentimental positivo que hay en el texto del mensaje.
- **fe_num_neut_emojis**: indica el número de emoticonos con un valor sentimental neutro que hay en el texto del mensaje.
- **fe_num_neg_emojis**: indica el número de emoticonos con un valor sentimental negativo que hay en el texto del mensaje.
- **fe_sentiment_avg**: esta columna indica el valor promedio del "sentiment score" de cada emoticono que hay en el texto. Este valor se obtiene del "Emoji Sentiment Ranking" [23] donde cada emoticono tiene un valor entre -1 y 1.

  Por ejemplo, en la siguiente frase:

  "Hey there... Life's been rough lately, and I'm really struggling. It's like a dark cloud is hanging over me all the time 😕. I just needed to vent and let you know how I'm feeling. Thanks for being there ♥♥♥"

  En esta oración tenemos 4 emoticonos, de los cuáles tres son positivos y uno es negativo. Si buscamos estos emoticonos en la tabla de Emoji Sentiment Ranking obtenemos los siguientes valores de sentiment score para cada uno:

  - 😕: 0.657
  - ♥: -0.522

  Si realizamos el promedio nos sale el siguiente valor para **fe_sentiment_avg**:

  $$\frac{0{,}657 * 3 + (-0{,}522)}{4} = 0{,}362 \tag{3.2}$$

  De esta forma, calcularíamos ese valor para todos los textos, en caso de que tengan emoticonos.
- **message_without_emojis**: el texto del mensaje sin emoticonos y sin emoticonos en formato texto.
- **emojis**: en esta columna podemos ver una concatenación de todos los emoticonos que aparecen en el mensaje.

Estas columnas la hemos añadido para poder extraer posteriormente características sobre ellas. Con la adición de las columnas relacionadas con los emoticonos, podemos añadir peso a la polaridad del mensaje en función de si estos son más positivos

o negativos. Además, añadiendo la columna del mensaje sin emoticonos podríamos ver la diferencia de polaridad entre los mensajes con emoticonos y sin emoticonos.

La columna que indica el número de palabras en mayúscula también nos puede servir para observar si las palabras en mayúscula afectan a la polaridad y el sentimiento de un mensaje. Podría ser posible que los mensajes que conlleven más palabras en mayúscula sean aquellos en los que el usuario escribe con ira o sorpresa.

Con todas estas columnas nuevas tenemos nuestros ficheros nuevos con los que hemos trabajado y que hemos usado para realizar las traducciones que se explican en el siguiente apartado.

3.1.2. Traducción de los textos

Como ya sabemos, los mensajes proporcionados por la organización están en español, pero el sistema que hemos usado para participar en la tarea es un sistema que solo funciona con textos en inglés. Por tanto, en vez de adaptar el sistema al español se propuso la idea de traducir los textos a español. Esta fue la tarea de la que yo me encargué.

Para traducir los textos, se usaron varios métodos con diferentes resultados, que posteriormente se compararon para ver qué forma de traducción era la más adecuada. Las diferentes formas para traducir los mensajes que se barajaron fueron las siguientes:

- **Google Translator**: una de las formas con las que se tradujeron los mensajes, fue con la API de Google Translator Ajax API. Concretamente, se usó la biblioteca oficial que tienen para el lenguaje de programación Python: googletrans [24].

  Esta es la opción que finalmente se utilizó para entrenar el modelo. La API es bastante rápida y el resultado de la traducción de los mensajes es muy bueno.

  En las siguientes imágenes vamos a ver el script para la traducción y el uso de la API de Google Translator a través de la biblioteca googletrans de Python:

```
1 from googletrans import Translator
2
3 translator = Translator()
```

Listado 3.1: Importar biblioteca y crear objeto googletrans

  Primero tenemos que importar la biblioteca y crear el objeto que vamos a usar para traducir los textos. Este código podemos verlo en el Código 3.1.

```
with open(nombre_fichero, newline="", encoding="utf-8") as csvfile:
    datos_csv = csv.reader(csvfile, delimiter=";")

    for fila in datos_csv:
        lista_datos.append(fila)
```

Listado 3.2: Importación del conjunto de datos

Como apreciamos en el Código 3.2, usando la biblioteca csv importamos nuestro fichero que contiene el conjunto de datos. Los parámetros que se usan para importarlos están adaptados a nuestro fichero (por ejemplo, el delimitador y la forma en que está codificado nuestro fichero). Cada fila se almacena como elemento en un array.

Para trabajar bien con este conjunto de datos, por cada mensaje crearemos un diccionario de Python donde almacenaremos sus datos. Las claves serán los nombres de las columnas y el valor de cada clave será el correspondiente al de la fila del mensaje. Aunque en el Código 3.3 no se muestra, también se elimina el primer elemento del array que creamos anteriormente porque contiene los títulos de las columnas y eso no cuenta como mensaje.

```
messages = []
for fila in lista_datos:
    message = {}
    for i in range(len(variables)):
        message.update({variables[i]: fila[i]})
    messages.append(message)
```

Listado 3.3: Almacenamiento de los mensajes

Una vez tenemos todos los mensajes preparados para traducirlos, creamos la función a la que vamos a llamar cada vez que vayamos a traducir un mensaje. Se usa el objeto de traducción que hemos creado al principio, pasándole como parámetros el mensaje que queremos traducir y el lenguaje destino que queremos obtener (que en nuestro caso es inglés y se indica como “en”).

```
def translation(message):
    traduccion = translator.translate(message, dest="en")
    return traduccion
```

Listado 3.4: Función para traducir un texto de español a inglés

En el Código 3.4 tenemos lo explicado anteriormente.

Finalmente, recorremos nuestro array con todos los mensajes llamando a la función anterior para traducir las columnas de “message”, “message_without_emojis” y “message_emojized”. En el Código 3.5 se muestra como se realiza este proceso.

```
    with open(fichero_traducido, "w", newline="", encoding="utf-8") as csvfile:
        encabezados = messages[0].keys()
        csv_writer = csv.DictWriter(csvfile, fieldname=encabezados)
        csv_writer.writeheader()

        cont = 0
        for message in messages:

            traduccion = translation(messages["message"])
            messages["message"] = traduccion.text

            traduccion = translation(messages["message_without_emojis"])
            messages["message_without_emojis"] = traduccion.text

            traduccion = translation(messages["message_emojized"])
            messages["message_emojized"] = traduccion.text
```

Listado 3.5: Traducción de todos los textos de cada mensaje

- **OpenAI API**: otra opción con la que también se tradujeron los textos fue con ChatGPT. ChatGPT es un gran avance en el campo de la Inteligencia Artificial y es una plataforma que no deja de ser Procesamiento del Lenguaje Natural. Por tanto, se pensó en traducirlos con esta herramienta. Pero traducirlos desde la aplicación web es demasiado lento porque son muchos mensajes, por tanto, se ha usado también la API oficial de OpenAI. Esta API te permite enviar prompts usando diferentes modelos pre entrenados. Nosotros usamos concretamente, el modelo text-davinci-003 [25]. Se eligió este modelo para realizar las llamadas a la API porque es un modelo que permite una mayor entrada de texto (los prompts de OpenAI funcionan con tokens) y es un modelo que trabaja muy bien con tareas relacionadas con el lenguaje. Como nuestra tarea es la traducción, pues finalmente se decidió que era el modelo más apropiado para nuestro objetivo.

- **Modelo pre entrenado de Hugging Face**: esta opción es una de las que peores resultados daba. El modelo que usó fue el modelo **Helsinki-NLP/opus-mt-es-en** [26] que es un modelo basado en transformers. Los transformers son una arquitectura basada en aprendizaje profundo. Son modelos que permiten realizar tareas mientras el modelo se entrena en paralelo, utilizando como potenciador el mecanismo de atención.

  Las principales desventajas de este modelo fueron que era muy lento, ya que las respuestas a las llamadas a la API eran muy lentas y el resultado de estas llamadas era bastante malo. Además, las entradas de texto para los prompts tienen un máximo de caracteres muy limitado (512 tokens). El script que se realizó también fue realizado en Python usando la biblioteca de “transformers”, desde la que descargamos el modelo que queremos. También, se usó NLTK para tokenizar los textos de más de 512 tokens en diferentes para poder traducir las oraciones por

separado y luego juntarlas. Esto daba problemas de contexto porque al separar un texto en diferentes oraciones, se puede perder el contexto del texto.

- **Deepl Translator**: esta opción es una de las más potentes de todas, el problema está en que la API de deepl [27] requiere de un pago para aumentar el máximo de tokens permitidos por mes. Con una sola traducción del conjunto de datos de entrenamiento se consumían todos los tokens permitidos por mes. Por tanto, por este motivo esta opción para traducir los textos fue descartada finalmente.

3.1.3. Ingeniería de características (Feature Engineering)

Para hablar de este apartado, primero vamos ver una pequeña introducción a la ingeniería de características.

Una característica es una representación numérica de datos en bruto. Una característica puede ser cualquier cosa, ya que esta se puede representar de forma numérica de muchas formas diferentes. Estas características deben obtenerse a partir de los tipos de datos que tenemos nosotros. También decir que el modelo debe ser apropiado para las características que tengamos. Es esencial elegir adecuadamente las características más interesantes y con potencial para nuestro modelo.

Este proceso de elegir las características es lo que llamamos ingeniería de características. En este proceso tratamos de formular las características más adecuadas en función de los datos que tenemos, el modelo que vamos a usar y la tarea en cuestión que vamos a tratar.

Es muy importante el número de características que vamos a usar. Tener pocas características puede ser insuficiente para nuestro modelo, por lo que no podrá realizar la tarea que queremos. Y si, por el contrario, tenemos un exceso de características el modelo será más difícil de entrenar y mucho más costoso computacionalmente. Además, si la gran parte de estas características son irrelevantes, el modelo será también un modelo malo para nuestra tarea.

Nuestro objetivo es crear un archivo en formato CSV que vamos a pasar a nuestro script de autoML, que se encargará de elegir las mejores características de ese archivo para usar en el entrenamiento del modelo. El autoML, tal y como su nombre indica, es el aprendizaje automático automatizado. Gracias a la biblioteca Pycaret de Python [28] podemos realizar el proceso de entrenamiento de los modelos de forma automática sin necesidad de crear un script a mano para esto.

Por tanto, nosotros lo que debemos hacer es añadir las características que nosotros consideremos más adecuadas al fichero. En nuestro caso, vamos a añadir las siguientes características, que además ilustraré con fragmentos de código del script que se ha usado para este proceso.

Para realizar este proceso, vamos a trabajar con DataFrames, que son un tipo de estructura de datos que nos permitirá trabajar con datos tabulares, es decir, como si fuese una tabla de Excel con filas y columnas. Cada columna tiene un índice que indica el tipo de valor que hay en esa columna.

```
train_df = pd.read_csv('models/google_dataset/task_2a_train_en.csv', sep=';', encoding='
    latin-1')
train_df['type'] = 'train'
trial_df = pd.read_csv('models/google_dataset/task_2a_trial_en.csv', sep=';', encoding='
    latin-1')
trial_df['type'] = 'trial'

erisk_df_no_blank_posts = pd.concat([train_df, trial_df], axis=0)
erisk_df_no_blank_posts.dropna(subset=['message_without_emojis'], inplace=True)
erisk_df_no_blank_posts.reset_index(drop=True, inplace=True)
erisk_df_no_blank_posts.rename(columns={'label': 'Class'}, inplace=True)
erisk_df_no_blank_posts.rename(columns={'message_without_emojis': 'Text'}, inplace=True)
```

Listado 3.6: Importación de los conjuntos de datos traducidos

Como se aprecia en el Código 3.6, podemos ver el proceso de importación de los ficheros (tanto entrenamiento como prueba) y posteriormente la creación del DataFrame concatenando estos dos ficheros. Después, simplemente eliminamos aquellas filas que tengan valor nulo en la columna “message_without_emojis”, reseteamos los índices, le cambiamos el nombre a un par de columnas y listo, ya podemos empezar a trabajar con nuestro DataFrame.

A partir de aquí, vamos a ir añadiendo columnas al DataFrame con valores que vamos a calcular para cada fila, es decir para cada mensaje. Estas columnas serán características.

Análisis empático (Empath Analysis)

Lo primero que vamos a hacer es realizar un análisis empático. Para ello vamos a usar la biblioteca empath de Python [29]. Esta biblioteca sirve para realizar análisis de sentimientos y emociones en texto, identificando temas y categorías asociadas con el contenido del texto. También vamos a usar un lematizador para reducir las palabras a su forma base o lema.

```
lemmatizer = WordNetLemmatizer()

erisk_df_no_blank_posts['cleaned_text'] = erisk_df_no_blank_posts['Text'].progress_apply(
    clean_text)

empath = Empath()
for empath_ in ['alcohol','hate','envy','health','nervousness','weakness','horror','
    suffering','kill','fear','friends','sexual','body','family','irritability','violence'
    ,'sadness','disgust','exasperation','emotional','anger','poor','pain','timidity','
    cheerfulness','medical_emergency','ragedfc','positive_emotion','negative_emotion','
    ugliness','weapon','shame','torment','help', 'office', 'sleep', 'money', 'school','
    home', 'hygiene', 'phone', 'work', 'appereance', 'optimism', 'youth', 'joy', '
    white_collar_job','morning', 'night', 'college', 'sports', 'neglect','disappointment'
    , 'children', 'contentment','music', 'musical', 'deception', 'blue_collar_job', '
    clothing', 'valuable', 'swearing_terms', 'exercise']:
    erisk_df_no_blank_posts[empath_] = erisk_df_no_blank_posts['cleaned_text'].
        progress_apply(lambda x: empath.analyze(lemmatizer.lemmatize(str(x)), categories
        =[empath_])[empath_])
erisk_df_no_blank_posts.drop(['cleaned_text'], axis=1, inplace=True)
```

Listado 3.7: Análisis empático de los mensajes

Como vemos en el Código 3.7 anterior, el proceso es un bucle en el que por cada categoría del array que está indicado ahí (alcohol, hate, envy, . . .) vamos a calcular su valor dentro de cada mensaje. Se lematiza el mensaje, y se analiza con cada categoría. Después obtenemos el valor del análisis y lo almacenamos en una nueva columna en el DataFrame con el nombre de esa categoría. De esta forma para cada mensaje realizamos un análisis de cada una de las categorías indicadas y lo almacenamos en su columna correspondiente.

Hay que decir que justo antes del bucle, creamos una columna con el título “cleaned_text”, donde guardamos el mensaje habiendo eliminado URLs, hashtags, signos de puntuación, palabras con menos de 2 caracteres y finalmente dejando todo en minúsculas. Todo esto lo hacemos llamando a una función auxiliar justo antes de entrar al bucle.

Parte del discurso (Part of Speech)

En esta parte vamos a etiquetar cada palabra del mensaje con su categoría gramatical (adjetivos, adverbios, verbos y sustantivos). Esta parte es muy importante dentro de la selección de características para poder entender la estructura gramatical y el significado de las oraciones que tenemos. El conocimiento de las partes del discurso puede ser muy útil para extraer las características relevantes de los datos.

Después de etiquetar cada palabra de cada mensaje vamos a realizar el conteo de cada categoría y vamos a almacenar ese valor en una nueva columna que será esa

categoría gramatical.

```
erisk_df_no_blank_posts['cleaned_text'] = erisk_df_no_blank_posts['Text'].progress_apply(
    clean_text_no_lower)

erisk_df_no_blank_posts['fe_pos_adjs'] = get_pos_tags(tag='ADJ')
erisk_df_no_blank_posts['fe_pos_advs'] = get_pos_tags(tag='ADV')
erisk_df_no_blank_posts['fe_pos_verbs'] = get_pos_tags(tag='VERB')
erisk_df_no_blank_posts['fe_pos_nouns'] = get_pos_tags(tag='NOUN')
```

Listado 3.8: Parte del discurso de cada mensaje 1

En el Código 3.8 se ve cómo almacenamos los valores que extraemos de la función auxiliar en las columnas nuevas. A esta función le pasamos como parámetro la etiqueta de la que queremos realizar el conteo para ir comparando con cada palabra dentro de la función.

```
def get_pos_tags(tag):
    pos = []
    for doc in tqdm(nlp.pipe(erisk_df_no_blank_posts['cleaned_text'].values, batch_size
        =16), total=erisk_df_no_blank_posts.shape[0]):
        pos_list = [token.pos_ for token in doc if token.pos_ == tag]
        pos.append(len(pos_list))
    return pos
```

Listado 3.9: Función auxiliar 1 para obtener el valor de una etiqueta concreta

En el Código 3.9, tenemos la función auxiliar que utilizamos para el Código 3.8. Como vemos recorremos cada fila del DataFrame y realizamos el conteo con la columna “cleaned_text” como hicimos con el Análisis Empático. Después almacenamos en una lista todas aquellas palabras cuya categoría coincide con la categoría que nosotros hemos indicado en la llamada de la función. Finalmente devolvemos el tamaño de esa lista y ese valor será el que almacenemos en la nueva columna.

A parte de las categorías anteriores, también hemos realizado el conteo de las palabras cuya categoría gramatical sean: verbos en pasado, superlativos y cuantificadores. Las dos primeras categorías se han contado de la misma forma que las cuatro anteriores.

```
erisk_df_no_blank_posts['fe_pos_past_tense_verbs'] = get_pos_fine_grained(tag=['VBD'])

erisk_df_no_blank_posts['fe_pos_superlative'] = get_pos_fine_grained(tag=['JJS', 'RBS'])
```

Listado 3.10: Parte del discurso de cada mensaje 2

La función con la que se han calculado sí que es diferente, pero solo cambia en el tipo de parámetro que se pasa a esta. Antes pasábamos un string y comparábamos con ese string y ahora pasamos una lista y miramos si la categoría está dentro de esa

lista.

```
def get_pos_fine_grained(tag):
    erisk_df_no_blank_posts['cleaned_text'] = erisk_df_no_blank_posts['Text'].apply(
        lambda x: clean_text(str(x)))
    pos = []
    for doc in tqdm(nlp.pipe(erisk_df_no_blank_posts['cleaned_text'].values, batch_size
        =16), total=erisk_df_no_blank_posts.shape[0]):
        pos_list = [token.pos_ for token in doc if token.tag_ in tag]
        pos.append(len(pos_list))
    return pos
```

Listado 3.11: Función auxiliar 2 para obtener el valor de una etiqueta concreta

En el Código 3.11 podemos ver la función auxiliar de la que he hablado.

Por último, tenemos los cuantificadores. El valor de esta categoría gramatical sí que se ha obtenido de una forma diferente. Se han usado expresiones regulares y la biblioteca Regex de Python para ello. Hemos creado una lista con algunos de los cuantificadores más usados en inglés y hemos usado la función "findall" de Regex donde pasamos todas las expresiones que hemos almacenado en una lista, pero unidas en un string separadas por el signo "|", y el string donde queremos buscar las ocurrencias, en este caso será cada mensaje.

En el Código 3.12 podemos ver la lista de cuantificadores y cómo calculamos su valor para cada uno.

```
quantifiers_list = ['some', 'several', 'a␣number␣of', 'enough', 'numerous', 'plenty␣of',
    'a␣lot␣of', 'lots␣of', 'much', 'many', 'few', 'little']
erisk_df_no_blank_posts['fe_quantifiers'] = erisk_df_no_blank_posts['Text'].
    progress_apply(
    lambda x: len(re.findall(r"|".join(regex for regex in quantifiers_list), clean_text(
        str(x)))))
```

Listado 3.12: Obtención del valor para los cuantificadores

Análisis de sentimientos: Subjetividad y polaridad

Tratándose de un modelo de detección de depresión, es necesario realizar un análisis de sentimientos. Para ello usamos VADER (Valence Aware Dictionary and Sentiment Reasoner). VADER es una herramienta de análisis de sentimientos diseñada para trabajar con texto en inglés. Destaca por su capacidad para manejar expresiones emocionales en texto, incluso aquellas que son comúnmente desafiantes para otras herramientas de análisis de sentimientos.

El funcionamiento de VADER de una forma general es el siguiente:

1. **Diccionario de polaridad y reglas heurísticas**: VADER utiliza un diccionario previamente construido que asigna puntuaciones de polaridad (positiva, negativa o neutra) a palabras en inglés. Estas puntuaciones están predefinidas y reflejan la valencia de las palabras.

   Además, VADER incorpora reglas heurísticas para manejar casos especiales, como el uso de mayúsculas, signos de exclamación y emojis.

2. **Puntuación compuesta**: la puntuación compuesta de VADER es el resultado de combinar las puntuaciones de polaridad de las palabras de una oración, teniendo en cuenta sus intensidades y la presencia de ciertos elementos gramaticales (por ejemplo, la presencia de negaciones).

   La puntuación compuesta proporciona una medida general del sentimiento de la oración.

3. **Puntuaciones individuales**: VADER también genera puntuaciones individuales para la polaridad y la intensidad.

4. **Resultados en formato diccionario**: VADER devuelve los resultados en forma de diccionario, que incluye la puntuación compuesta, así como las puntuaciones individuales para positividad, negatividad y neutralidad.

En nuestro caso, vamos a obtener las puntuaciones compuestas de VADER y, para ello, haremos uso de la biblioteca vaderSentiment de Python [30]. Crearemos un objeto analizador de sentimientos de VADER y usaremos la función “polarity_scores”, a la que le pasaremos el string del mensaje y la polaridad de la que queremos obtener la puntuación. En el Código 3.13 podemos ver el cálculo de estos valores.

```
1 sid_obj = SentimentIntensityAnalyzer()
2 erisk_df_no_blank_posts['fe_vader_positive_sentiment'] = erisk_df_no_blank_posts['Text'].
      progress_apply(lambda x: sid_obj.polarity_scores(clean_text_no_lower(str(x)))['pos'])
3 erisk_df_no_blank_posts['fe_vader_neutral_sentiment']  = erisk_df_no_blank_posts['Text'].
      progress_apply(lambda x: sid_obj.polarity_scores(clean_text_no_lower(str(x)))['neu'])
4 erisk_df_no_blank_posts['fe_vader_negative_sentiment'] = erisk_df_no_blank_posts['Text'].
      progress_apply(lambda x: sid_obj.polarity_scores(clean_text_no_lower(str(x)))['neg'])
```

Listado 3.13: Análisis de sentimientos con VADER

Para realizar un cálculo general de la subjetividad y la polaridad de la oración vamos a usar la biblioteca TextBlob [31]. TextBlob es una biblioteca de Procesamiento del Lenguaje Natural en Python que simplifica muchas tareas comunes de procesamiento de texto. Está construida sobre la biblioteca NLTK y proporciona una interfaz fácil de usar para realizar operaciones de procesamiento de texto, que en nuestro caso sería análisis de sentimientos.

Ahora vamos a pasar a ver los fragmentos de código de cada parte. A continuación, tenemos la obtención de los valores de polaridad y subjetividad con TextBlob.

```
erisk_df_no_blank_posts["fe_subjectivity"] = erisk_df_no_blank_posts["Text"].
    progress_apply(lambda x:TextBlob(clean_text_no_lower(str(x))).sentiment.subjectivity)
erisk_df_no_blank_posts["fe_polarity"]= erisk_df_no_blank_posts["Text"].progress_apply(
    lambda x:TextBlob(clean_text_no_lower(str(x))).sentiment.polarity)
```

Listado 3.14: Análisis de sentimientos con TextBlob

Como vemos en el Código 3.14, simplemente obtenemos la subjetividad y polaridad de cada mensaje de la parte de análisis de sentimientos de TextBlob.

Para terminar esta parte de análisis de sentimientos, también vamos a realizar un análisis usando un transformer. Hemos hecho uso de un modelo basado en transformers para obtener una puntuación para cada mensaje de polaridad, igual que hemos hecho con VADER. De esta forma, vamos a probar dos formas diferentes de medir la polaridad de los mensajes.

El modelo que hemos usado es un modelo de Hugging Face que se llama **Twitter-roBERTa-base for Sentiment Analysis** [32]. Este modelo analiza un texto y da una puntuación para tres categorías diferentes: positivo, negativo y neutral. Para hacer uso de este modelo necesitamos usar las bibliotecas de Python transformers [33] y pipeline [34]. Estas bibliotecas permiten usar los modelos de Hugging Face indicando el dispositivo que quieres utilizar para realizar el análisis.

En mi caso, al no disponer de una tarjeta gráfica integrada NVIDIA, no dispongo de CUDA, que es el software que usa PyTorch para ejecutar los modelos basados en transformers, por tanto, tengo que indicar que use la CPU para este proceso.

```
tokenizer = AutoTokenizer.from_pretrained(cardiffnlp/twitter-roberta-base-sentiment-
    latest)
model = AutoModelForSequenceClassification.from_pretrained(cardiffnlp/twitter-roberta-
    base-sentiment-latest)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
model.device

torch.cuda.empty_cache()
gc.collect()

with torch.no_grad():
    erisk_df_no_blank_posts["fe_sentiment"] = erisk_df_no_blank_posts["Text"].
        progress_apply(lambda x: model(**tokenizer(preprocess(str(x).replace("\n", "␣").
        replace("\t", "␣")),truncation=True, padding=True, max_length=511, return_tensors
        ='pt').to(device)))

erisk_df_no_blank_posts["fe_sentiment"] = erisk_df_no_blank_posts["fe_sentiment"].
    progress_apply(lambda x: softmax(x[0][0].cpu().detach().numpy()))
```

```
14
15 emotions = ['negative', 'neutral', 'positive']
16 for i, emotion in enumerate(emotions):
17     erisk_df_no_blank_posts['fe_roberta_base_sentiment_' + emotion] =
           erisk_df_no_blank_posts['fe_sentiment'].progress_apply(lambda x: x[i])
```

Listado 3.15: Análisis de sentimientos con Transformers

Como se aprecia en el Código 3.15, primero realizamos el análisis para cada mensaje y posteriormente añadimos cada uno de los valores de cada categoría a su columna del DataFrame.

Comentarios tóxicos

Otras características que vamos a analizar tienen que ver con la toxicidad. Para eso vamos a usar otro modelo de Hugging Face basado en transformers que se llama **unitary/toxic-bert**. Este modelo analiza un texto y da una puntuación a diferentes categorías de toxicidad: toxic, severe_toxic, obscene, threat, insult y identity_hate.

El procedimiento es el mismo que para los pasos anteriores, le pasamos los mensajes al modelo para cada mensaje dará una puntuación para cada una de las categorías mencionadas antes. Esto se puede ver en el Código 3.16

```
1  dv = torch.device("cuda" if torch.cuda.is_available() else "cpu")
2  pipe = pipeline(model="unitary/toxic-bert", device=dv, return_all_scores=True)
3
4  batch_size = 8
5  sequences = erisk_df_no_blank_posts['Text'].apply(lambda x: preprocess(str(x).replace("\n
       ", "␣").replace("\t", "␣"))).to_list()
6  results = []
7  with torch.no_grad():
8      for i in tqdm(range(0, len(sequences), batch_size)):
9          results += pipe(sequences[i:i+batch_size], padding=True, truncation=True)
10
11 erisk_df_no_blank_posts['fe_toxicity'] = results
12 erisk_df_no_blank_posts['fe_toxic'] = erisk_df_no_blank_posts['fe_toxicity'].apply(lambda
       x: x[0]['score'])
13 erisk_df_no_blank_posts['fe_severe_toxic'] = erisk_df_no_blank_posts['fe_toxicity'].apply
       (lambda x: x[1]['score'])
14 erisk_df_no_blank_posts['fe_obscene'] = erisk_df_no_blank_posts['fe_toxicity'].apply(
       lambda x: x[2]['score'])
15 erisk_df_no_blank_posts['fe_threat'] = erisk_df_no_blank_posts['fe_toxicity'].apply(
       lambda x: x[3]['score'])
16 erisk_df_no_blank_posts['fe_insult'] = erisk_df_no_blank_posts['fe_toxicity'].apply(
       lambda x: x[4]['score'])
17 erisk_df_no_blank_posts['fe_identity_hate'] = erisk_df_no_blank_posts['fe_toxicity'].
       apply(lambda x: x[5]['score'])
```

Listado 3.16: Análisis de comentarios tóxicos

Signos de puntuación

Una característica que puede ser interesante analizar es el número de signos de puntuación que tiene cada mensaje. Esto lo hacemos con la biblioteca regex de Python que es para expresiones regulares.

```
punctuation_signs = r"[\?¿!¡]"
erisk_df_no_blank_posts["fe_punct_signs"] = erisk_df_no_blank_posts["Text"].
    progress_apply(lambda x: len(re.findall(punctuation_signs, str(x))))
```

Listado 3.17: Análisis de los signos de puntuación

Tal y como vemos en el Código 3.17, primero definimos la expresión regular para buscar signos de puntuación y, después, para cada mensaje llamamos a la función "findall" para contar todos los signos que hemos definido en la expresión regular anterior se encuentran en cada mensaje.

Emoticonos

El número de emoticonos también puede ser una característica significativa que podría indicar si el mensaje es depresivo o no. Para ello usamos la biblioteca emot de Python [35]. Esta biblioteca tiene los mismos datos que el Emoji Sentiment Ranking que usamos para añadir las columnas sobre la información de los emoticonos en el conjunto de datos. Esta biblioteca es muy cómoda para este proceso, solo necesitamos importar las listas de emoticonos (emoticonos escritos con símbolos y emoticonos de imagen) y por cada mensaje buscamos si hay alguno dentro de las listas.

```
erisk_df_no_blank_posts['fe_num_emojis_emoticons'] = erisk_df_no_blank_posts['Text'].
    progress_apply(lambda x: len([x_aux for x_aux in str(x).split('␣') if x_aux in list(
    EMOTICONS_EMO.keys()) + list(UNICODE_EMOJI.keys())]))
```

Listado 3.18: Análisis de los emoticonos

Como vemos en el Código 3.18, al final, el dato que almacenamos en la nueva columna que se añadirá al DataFrame será el número total de emoticonos que hay dentro del mensaje.

Negaciones

Para esta parte lo que se hizo fue recopilar en una lista negaciones típicas en inglés. Se juntan todas en una expresión regular y se realiza lo mismo que en la sub-

sección relacionada con los Signos de Puntuación.

```
neg_forms =["Abrasive","Apathetic","Controlling","Dishonest","Impatient","Anxious","
    Betrayed","Disappointed","Embarrassed","Jealous","Abysmal","Bad","Callous","Corrosive
    ","Damage","Despicable","D o n t ","Dont","Enraged","Fail","Gawky","Haggard","Hurt","
    Icky","Insane","Jealous","Lose","Malicious","Naive","Not","Objectionable","Pain","
    Questionable","Reject","Rude","Sad","Sinister","Stuck","Tense","Ugly","Unsightly","
    Vice","Wary","Yell","Zero","Adverse","Banal","C a n t ","Corrupt","Damaging","
    Detrimental", ...]
neg_forms_lit = r"|".join("\\b"+word.lower()+"\\b" for word in neg_forms)
erisk_df_no_blank_posts['fe_num_negations'] = erisk_df_no_blank_posts['Text'].
    progress_apply(
    lambda x: len(list(set(re.findall(neg_forms_lit, clean_text(str(x)))))))
```

Listado 3.19: Análisis de las negaciones

Al igual que en el resto de características, el valor que obtenemos y almacenamos es el número total de negaciones por cada mensaje.

Como se aprecia en el Código 3.19, al final de la lista hay unos puntos suspensivos. Esto quiere decir que la lista es mucho más extensa en el código real, por lo que he dejado solo algunas de las palabras simplemente para ver que se vea el tipo de palabras que son.

Número de palabras en mayúscula

Otra característica que obtenemos es el número de palabras en mayúscula por mensaje.

```
erisk_df_no_blank_posts['fe_number_uppercase_words'] = erisk_df_no_blank_posts['Text'].
    progress_apply(lambda x: len([word for word in str(x).split() if word.isupper()]))
```

Listado 3.20: Análisis de las palabras en mayúscula

Recorremos palabra por palabra de cada mensaje comprobando si está en mayúscula, como se aprecia en el Código 3.20. En caso de que así sea, sumamos uno al contador y finalmente guardamos ese valor en la nueva columna del DataFrame.

Número de letras en mayúscula

A diferencia del punto anterior, en este no contamos las palabras si no el número de letras en mayúscula. Recorremos el string que forma el mensaje letra por letra comprobando si la letra está en mayúscula. Al final del bucle almacenamos el valor del total de letras en mayúscula del mensaje en la nueva columna del DataFrame.

Legibilidad

En esta parte, vamos a realizar algunas medidas para comprobar la legibilidad del texto. Para ellos, haremos uso de la biblioteca readability de Python [36]. Esta biblioteca nos permite evaluar cómo de fácil es comprender un texto en función de diversos factores, como la longitud de las oraciones, la complejidad de las palabras y la estructura general del texto. Esta biblioteca puede ser muy útil en aplicaciones que desean analizar la facilidad de lectura de un contenido, como en la creación de material educativo, evaluación de textos para audiencias específicas o incluso en la mejora de usabilidad de sitios web.

Lo que vamos a hacer nosotros, es obtener unas medidas de legibilidad para cada mensaje del conjunto de datos. En nuestro caso, las medidas que hemos elegido son las siguientes:

- **Kincaid** [37]: este puntaje, también conocido como el puntaje Flesch-Kincaid, evalúa la facilidad de lectura de un texto en función de la longitud de las palabras y las oraciones. Cuanto más alto es el puntaje, más difícil de leer es el texto.
- **ARI (Automated Readability Index)** [38]: el Índice de Legibilidad Automatizado evalúa la complejidad de un texto en función de las palabras y las oraciones. Es similar al puntaje Kincaid, un puntaje más alto indica un texto más difícil.
- **Coleman-Liau** [39]: este método se basa en la cantidad de letras, palabras y oraciones para determinar la legibilidad. Al igual que los anteriores, cuanto más alto es el puntaje, más difícil es el texto.
- **Flesch Reading Ease** [40]: mide la facilidad de lectura en función de la longitud de las oraciones y de las palabras. Un puntaje más alto indica un texto más fácil de leer.
- **Gunning Fog Index** [41]: similar al puntaje Flesch-Kincaid, evalúa la dificultad de un texto en función de la longitud de las oraciones y la proporción de palabras complejas. Un puntaje más alto indica un texto más difícil.
- **LIX** [42]: LIX es una medida que tiene en cuenta la longitud de las palabras y la cantidad de frases largas en un texto. Cuanto más bajo es el puntaje, más fácil es el texto.
- **SMOG Index** [43]: el Índice SMOG evalúa la dificultad de un texto en función de las palabras polisilábicas. Un puntaje más alto indica un texto más difícil.

- **Dale-Chall Index** [44]: esta medida tiene en cuenta la cantidad de palabras difíciles en un texto. Se consideran palabras difíciles aquellas que no son comúnmente conocidas por los estudiantes de cuarto grado en los Estados Unidos. Un puntaje más alto índica un texto más difícil.

Todas estas métricas solo están disponibles para inglés, no se pueden usar si los textos de los que queremos evaluar la legibilidad están en español.

```
erisk_df_no_blank_posts['fe_kincaid_readability_index'] = erisk_df_no_blank_posts['Text'
    ].progress_apply(lambda x: readability.getmeasures(readability_preprocessing(
    clean_text_no_lower(str(x))), lang='en')['readability␣grades']['Kincaid'] if re.
    search(r"[a-zA-Z]", clean_text_no_lower(str(x))) != None else 0)
```

Listado 3.21: Análisis de legibilidad

Para esta parte, solo he incluido la obtención de una medida en el Código 3.21. El resto se calculan de la misma forma, pero cambiando el nombre de la métrica a evaluar.

Información de las oraciones

La biblioteca readability también nos sirve para evaluar diferentes aspectos de las oraciones y palabras en un texto. Para esta parte también la hemos usado para obtener los valores de las siguientes métricas de la parte de "sentence info" de readability:

- **Words**: con esta métrica obtenemos el número total de palabras en el texto.
- **Sentences**: indica la cantidad total de oraciones en el texto.
- **Paragraphs**: muestra la cantidad total de párrafos en el texto.
- **Long_words**: representa el número de palabras largas en el texto. La definición de "palabra larga" puede variar, pero a menudo se refiere a palabras que tienen un número significativo de sílabas.
- **Complex_words**: indica la cantidad de palabras consideradas como "complejas". La definición de "palabra compleja" también puede variar, pero a menudo se refiere a aquellas palabras que pueden ser difíciles de entender o que no son comunes.

Con estas métricas obtendremos información detallada sobre la estructura y complejidad del texto. Pueden ser características útiles para los modelos que probemos

ya que nos pueden ayudar a comprender la composición del texto. También son medidas útiles para comprender como influyen a las métricas de legibilidad que hemos obtenido en el apartado anterior, como las métricas Flesch, Kincaid, ARI, entre otras.

El código de esta parte es exactamente igual que el de la subsección anterior. Lo único que tenemos que hacer es sustituir el nombre de la métrica que queremos obtener.

Términos de medicamentos antidepresivos

Podría ser interesante analizar los nombres de los medicamentos antidepresivos por si encontramos alguno en algún mensaje. Esto nos podría indicar que la persona podría estar tomándolos. Al igual que hicimos con la subsección relacionada con las Negaciones, hemos recopilado una lista de medicamentos antidepresivos, concretamente los nombres de marcas y de químicos antidepresivos (recopilados por WebMD) disponibles en Estados Unidos.

Y al igual que en la subsección comentada, el código es igual. Simplemente recorremos palabra por palabra cada mensaje mirando si alguna de estas corresponde con alguno de los términos de la lista.

Información emocional

En este punto vamos a evaluar las emociones de los mensajes. En este caso hemos querido hacerlo de dos formas diferentes:

- Usando un modelo basado en transfomers, el cual es **cardiffnlp/roberta-base-emotion** [45]. Este modelo de Hugging Face es una versión del modelo "roberta-base" entrenado con el dataset de "tweet eval". El modelo proporciona una puntuación para cuatro emociones en un texto. Las emociones del texto que puntúa son: felicidad, optimismo, tristeza y enfado.

- Usando la **biblioteca NRCLex** de Python [46]. Esta biblioteca mide el afecto emocional del cuerpo de un texto. El diccionario de afecto contiene aproximadamente 27000 palabras y está basado en el diccionario de afecto del Consejo Nacional de Investigación de Canadá y la conjunto de sinónimos de WordNet de la biblioteca NLTK.

El proceso para ejecutar la primera opción es el mismo que usamos en el **Análisis de Sentimientos con Transformers**. Descargamos el modelo de Hugging Face que vamos a usar y le pasamos mensaje a mensaje para obtener la puntuación de las cuatro emociones en cada uno. Para cada emoción añadiremos una columna al DataFrame donde almacenaremos el valor de la puntuación obtenida de la emoción de cada mensaje.

Para la segunda opción, simplemente tenemos que pasarle el texto a la clase NRCLex y llamar a su función "affect_frequencies()". Está función nos proporcionará la puntuación para diversas emociones del texto. A continuación, en el Código 3.22 podemos ver cómo se realiza este proceso, donde por cada oración obtenemos puntuaciones para las siguientes emociones: miedo, enfado, previsión, confianza, sorpresa, positivo, negativo, tristeza, asco y felicidad.

```
1 erisk_df_no_blank_posts['fe_nrclex_emotions'] = erisk_df_no_blank_posts['Text'].
      progress_apply(lambda x: NRCLex(clean_text_no_lower(str(x)).replace("\n", "␣").
      replace("\t", "␣")).affect_frequencies)
2
3 for emotion in ['fear', 'anger', 'anticip', 'trust', 'surprise', 'positive', 'negative',
      'sadness', 'disgust', 'joy']:
4     erisk_df_no_blank_posts['fe_nrclex_emotion_' + emotion] = erisk_df_no_blank_posts['
          fe_nrclex_emotions'].progress_apply(lambda x: x[emotion])
```

Listado 3.22: Análisis de la información emocional con NRCLex

Información de metadatos: Fecha

Al examinar el conjunto de dato, podemos ver que cada mensaje viene fechado con el día, mes y año; y la hora (horas, minutos y segundos). Con esta información podemos obtener algunas características que podrían indicar si una persona tiene depresión, por ejemplo, puede ser que las personas que escriben por la noche tengan más síntomas de tener depresión que las personas que escriben por la mañana.

De esta forma, hemos definido cuatro intervalos de tiempo para ver en qué horas del día se escriben más mensajes. Los intervalos definidos son los siguientes:

- **Early_morning**: desde las 00:00 hasta las 05:00 am.
- **Morning**: desde las 05:00 hasta las 11:00 am.
- **Afternoon**: desde las 11:00 hasta las 21:00 pm.
- **Night**: desde las 21:00 hasta las 00:00 pm.

Para cada intervalo se creará una nueva columna en el DataFrame y, para clasificar cada mensaje, lo que hacemos es formatear la fecha con la biblioteca pandas de Python y comprobar en que intervalo horario se encuentra el mensaje. El intervalo en el que se encuentre el mensaje se marcará con un 1 y el resto quedarán marcados a 0.

Al igual que hemos clasificado los mensajes por intervalos horarios en un día también los hemos clasificado por meses. Igual que puede ser que una persona con síntomas de depresión es más probable que pueda escribir mensajes depresivos en un horario del día concreto, también puede haber épocas del año donde las personas que sufren depresión escriban más mensajes de este tono.

Por tanto, dividiremos los meses del año en cuatro intervalos:

- **First_season**: del mes 1 al mes 3.
- **Second_season**: del mes 3 al mes 6.
- **Third_season**: del mes 6 al mes 9.
- **Fourth_season**: del mes 9 al mes 12.

Para clasificarlos realizamos el mismo proceso realizado con los intervalos horarios.

A continuación, podemos ver un ejemplo para la clasificación del primer intervalo horario en el Código 3.23.

```
erisk_df_no_blank_posts['fe_posted_early_morning'] = erisk_df_no_blank_posts['Date'].
    progress_apply(lambda x: 1 if pd.to_datetime(x).hour < 5 and pd.to_datetime(x).hour
    >= 0 else 0)
```

Listado 3.23: Análisis de los intervalos horarios

Y aquí, un ejemplo del primer intervalo mensual en el Código 3.24.

```
erisk_df_no_blank_posts['fe_posted_first_season'] = erisk_df_no_blank_posts['Date'].
    progress_apply(lambda x: 1 if pd.to_datetime(x).month <= 3 and pd.to_datetime(x).
    month >= 1 else 0)
```

Listado 3.24: Análisis de los intervalos mensuales

Pronombres en primera persona

Otra característica que vamos a obtener es el número de veces que aparecen pronombres en primera persona. Esto lo hacemos porque nosotros queremos saber si la persona que ha escrito el mensaje es la que tiene síntomas de depresión. Por tanto, es importante saber si la persona que lo ha escrito está hablando en primera persona, ya que podría estar hablando de temas relacionados con la depresión, pero podría estar refiriéndose a otra persona.

Como el modelo funciona para textos en inglés los pronombres que vamos a buscar son: “i”, “me”, “my” y “mine”. Formamos una lista de expresiones regulares con estos pronombres y por cada mensaje contamos la cantidad total de pronombres que aparecen. Este valor será el almacenado en la nueva columna del DataFrame.

Términos o expresiones relacionados con la depresión

Para terminar esta parte de selección de características, hemos realizado una selección de expresiones regulares de diferentes categorías. Estas categorías son síntomas o estados ligados a la depresión extraídos del “Patient Health Questionnaire-9” [47]. De cada categoría hemos realizado una selección de expresiones regulares que pueden ser indicadores de tener ese síntoma o trastorno. El objetivo es dar un valor para cada una de las categorías, el cual será la suma total de las apariciones de cada una de las expresiones regulares en cada mensaje, es decir, para cada expresión tendremos el número de veces que aparece en el mensaje y una vez hemos calculado el valor de cada una, sumamos todos los valores y ese será el resultado para esa categoría.

Las categorías son las siguientes:

- **Anhedonia**: la anhedonia es un término utilizado en psicología y psiquiatría para describir la incapacidad de experimentar placer o disfrutar de las cosas que normalmente son gratificantes. Esto es diferente para cada persona, es decir, la anhedonia se manifiesta cuando una persona deja de sentir satisfacción por algo que antes si le producía satisfacción. Esto puede ir desde actividades sociales, a actividades de ocio o incluso con la comida.

- **Concentración**: la falta de concentración está asociada con la depresión de muchas formas. De hecho, es un criterio de diagnóstico de 3 tipos de depresión

(entre otros): Trastorno Depresivo Mayor, Trastorno Depresivo Persistente y Trastorno Disfórico Premenstrual.

- **Trastornos alimenticios**: los cambios en el comportamiento alimenticio son comunes en las personas que sufren de depresión, y estos cambios pueden manifestarse de diversas maneras. Tanto la pérdida de apetito como el aumento de apetito son síntomas que pueden estar asociados con la depresión. Estos cambios en el comportamiento alimenticio pueden tener consecuencias significativas en la salud física y mental de la persona. Estos cambios pueden estar asociados a la falta de autoestima de la persona y el cómo se percibe físicamente, y esto puede estar propulsado también por los estándares de belleza impuestos en la sociedad.

- **Cansancio o fatiga**: el cansancio o la fatiga son síntomas comunes de la depresión y a menudo están interrelacionados. La relación entre el cansancio y la depresión es compleja y puede manifestarse de diversas maneras. Por ejemplo, las personas que sufren de depresión a menudo informan una profunda sensación de cansancio o agotamiento. Aunque pueden dormir más de lo habitual, aún se sienten fatigadas y carecen de la energía necesaria para realizar actividades diarias.

- **Estado de ánimo**: la depresión afecta significativamente el estado de ánimo de una persona. Es un trastorno del estado de ánimo que va más allá de las variaciones normales en las emociones y puede tener un impacto profundo y persistente en el bienestar emocional. Algunas de las formas de las que puede afectar el estado de ánimo es tener una tristeza persistente, sentimiento de desesperanza, pérdida de interés y placer, irritabilidad, etc.

- **Cambios psicomotores**: los cambios psicomotores son alteraciones en la actividad física y en los procesos mentales que pueden estar asociados con diversos trastornos mentales, incluida la depresión. Estos cambios pueden manifestarse de diferentes maneras en las personas con depresión y están relacionados con la energía, la actividad y la velocidad del pensamiento.

- **Problemas de autoestima**: la autoestima se refiere a la evaluación subjetiva que una persona tiene sobre sí misma y esta puede influir en el desarrollo y la intensidad de la depresión, y a su vez, la depresión puede afectar a la autoestima de una persona de manera significativa. Las dos están relacionadas de forma bidireccional, es decir, afectan la una a la otra y viceversa.

- **Autolesiones**: las autolesiones son comportamientos deliberados de daño físico a uno mismo. Estos comportamientos pueden incluir, cortarse, quemarse,

golpearse o realizar otras acciones que causen daño físico con o sin intención de suicidarse. Las autolesiones a menudo coexisten con la depresión. Muchas personas que experimentan depresión también pueden recurrir a las autolesiones como una forma de hacer frente al dolor emocional, expresar emociones intensas o sentir alivio temporal.

- **Trastornos del sueño**: la relación entre la depresión y el trastorno del sueño es estrecha y bidireccional. La depresión puede afectar significativamente los patrones del sueño, y a su vez, los problemas de sueño pueden influir en el desarrollo de la depresión. Algunas formas de trastorno del sueño pueden ser: insomnio, hipersomnia, cambios en el ritmo circadiano, pesadillas y sueños perturbadores, etc.

- **Ataques de pánico**: la depresión y los ataques de pánico son dos trastornos mentales distintos, pero en algunas personas, pueden coexistir o estar interrelacionados. Por ejemplo, algunas personas pueden desarrollar los dos trastornos como respuestas anómalas al estrés. Las situaciones estresantes pueden desencadenar o exacerbar tanto la depresión como los ataques de pánico.

Como hemos dicho, hemos recogido un conjunto de expresiones de cada categoría que pueden ser indicios de ese trastorno. Para cada expresión contamos el número de veces que aparece en el mensaje y una vez buscadas todas las expresiones sumamos el número total de cada una como valor final para cada categoría.

Finalmente, habiendo obtenido los valores para todas las características que hemos querido obtener, transformamos el DataFrame con todos los valores a un fichero csv, que será el fichero con el que realizaremos los entrenamientos de los modelos que usaremos.

3.1.4. Fase de experimentación

Una vez terminada la fase de selección de características y obtenido nuestro fichero csv con todas estas, pasamos a la fase donde vamos a entrenar diferentes modelos y probar estos para ver qué resultados obtenemos de cada uno y valorar cuál es el que mejor funciona de todos.

Lo primero que hacemos en esta parte es agrupar ciertas características por nombre de usuario. Para ello, haremos suma o media dependiendo de qué tipo sean estas características:

- Para las características relacionadas con los sentimientos, las emociones, la toxicidad, la subjetividad o polaridad, y la legibilidad; realizaremos la media. Aplicamos la media con estas características porque son resultados probabilísticos o escalas continuas. Los valores probabilísticos pueden ir entre 0 y 1 algunos, si aplicamos la suma podemos superar los límites de los valores.
- Para el resto de características realizaremos la suma. Hacemos la suma con el resto de características porque representan recuentos o frecuencias. De esta forma, si sumamos, por ejemplo, el número de palabras en mayúscula de cada mensaje obtemos el número total de palabras en mayúscula que ha mandado un usuario.

Esto se hace porque

Como podemos ver en la Tabla 3.4, tanto las características de “TextBlob polarity” como “joy” son características ligadas los sentimientos y la polaridad. Por tanto, todos los resultados de esas características para un mismo usuario se agrupan realizando la media de todos los valores. Sin embargo, para la característica “Number of first person pronouns” realizamos la suma de todos sus valores.

Una vez realizada toda la organización y agrupación de todas las características, vamos a pasar a la parte de entrenamiento de los modelos y a la prueba de cada uno y a obtener los resultados.

SubjectID	TextBlob polarity	...	joy	...	Number of first person pronouns
1	-0.78	...	0.1	...	10
1	-0.81	...	0.08	...	5
1	-0.65	...	0.07	...	2
1	-0.1	...	0.15	...	0
1	0	...	0.17	...	1
1	0.12	...	0.19	...	1
1	0.87	...	0.81	...	3

A continuación, pasaremos a ver algunas de las herramientas utilizadas y la metodología que hemos seguido para entrenar los modelos.

SubjectID	TextBlob polarity	...	joy	...	Number of first person pronouns
1	-0.19	...	0.22	...	22

Tabla. 3.4: Ejemplo de la agrupación de características dependiendo del tipo

biblioteca de AutoML

Nuestro objetivo es probar diferentes modelos y comprobar cuál es el que mejor funciona. Para hacer este proceso más rápido y sacar un buen partido de las capacidades del aprendizaje automático, hemos utilizado la biblioteca PyCaret [28].

Esta biblioteca está diseñada para simplificar y acelerar el flujo de trabajo del aprendizaje automático. Está construida sobre bibliotecas como scikit-learn, XGBoost, LightGBM y otras. Su principal objetivo es permitir a los usuarios realizar tareas comunes de aprendizaje automático con un mínimo de código. Ofrece funciones para realizar el preprocesamiento de datos, como manejar valores perdidos, codificación de variables categóricas o escalamiento de características. También, permite la configuración automática de modelos mediante el uso de la función “setup()”. Esta función realiza la selección automática de modelos y ajusta los hiperparámetros utilizando técnicas como la búsqueda aleatoria. Concretamente, esta es la función que hemos utilizado nosotros en nuestro script de experimentación.

Validación-Cruzada K-Fold

Al tener un tiempo más limitado para crear el entrenamiento de los modelos se decidió aplicar una validación cruzada K-Fold. La validación cruzada K-Fold (K-Fold Cross Validation) es una técnica comúnmente utilizada en la evaluación de modelos de machine learning para obtener una estimación más robusta del rendimiento del modelo. En lugar de dividir el conjunto de datos en un solo conjunto de entrenamiento y otro de prueba, la validación cruzada divide los datos en k subconjuntos (folds) de aproximadamente igual tamaño. Luego, el modelo se entrena k veces, utilizando cada vez un fold diferente como conjunto de prueba y los folds restantes como conjunto de entrenamiento. En la siguiente imagen se muestra de manera más clara cómo funciona:

Como podemos ver en la Figura 3.2 se ha elegido una k igual a 5. Podemos ver que, como la k es 5, el conjunto de datos se ha dividido en 5 partes, y en cada iteración (“Performance” en la imagen) se selecciona una de esas partes como conjunto de

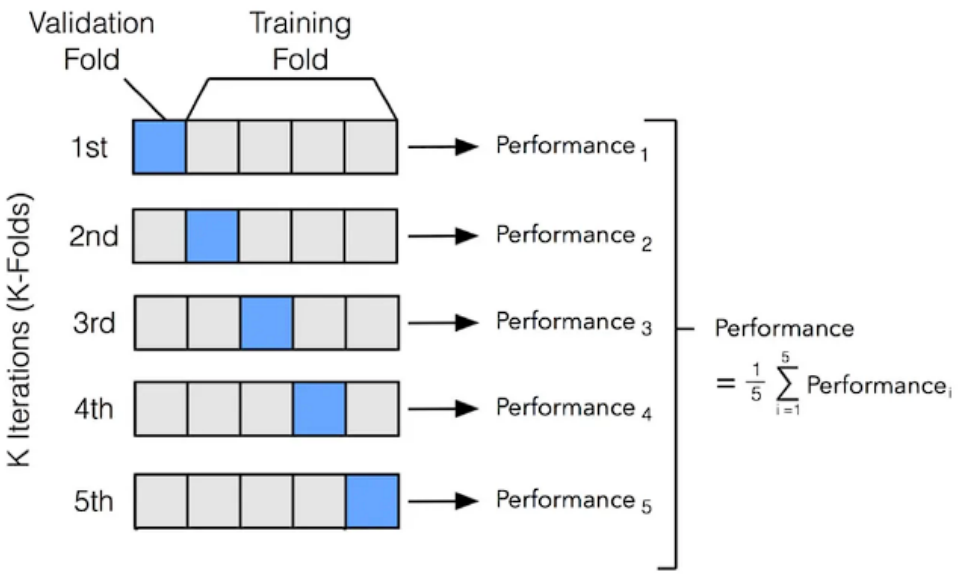

Figura 3.2: Ejemplo K-Fold Cross Validation.

prueba (el cuadrado coloreado en azul) y el resto de partes se usan como conjunto de entrenamiento. El resultado final será la media de resultados de todas las partes.

Esta manera de realizar el entrenamiento de los datos ayuda a abordar el problema de la variabilidad en la estimación del rendimiento del modelo que puede surgir cuando se utiliza una única división de datos en entrenamiento/prueba. Al realizar múltiples divisiones y promediar los resultados, se obtiene una estimación más robusta del rendimiento del modelo, ya que se evalúa en diferentes subconjuntos de datos.

Selección de características

Con el objetivo de potenciar la capacidad predictiva del modelo y disminuir la dimensionalidad del conjunto de datos, implementamos técnicas de selección de características. Concretamente, usamos una función de la biblioteca scikit-learn: “SelectFromModel”. Esta función nos permite seleccionar las características más importantes entrenando un estimador y extrayendo las características mejor clasificadas según su importancia. De esta forma, mejoraremos el rendimiento del modelo centrándonos en las características que sean más relevantes.

Selección del modelo

Una vez terminada la fase de selección de características, vamos a entrenar varios modelos de aprendizaje automático usando el flujo de trabajo de PyCaret. PyCaret admite una gran cantidad de algoritmos de diferentes tipos y nos da una facilidad de comparar su rendimiento en el conjunto de datos. Para elegir los modelos con los mejores resultados, los hemos ordenado en función del F1-score. Esta métrica se utiliza mucho en problemas de clasificación binaria como el que afrontamos nosotros. Con ella, podemos evaluar el equilibrio entre "precision" y "recall", con las que se calcula el F1-score. Después, se eligen los cuatro modelos más eficaces en función de esta métrica para evaluarlos y compararlos posteriormente.

Como resumen de la metodología usada, el proceso total consiste en utilizar el flujo de trabajo de PyCaret para preprocesar los datos, realizar la validación cruzada K-Fold, dividir el conjunto de datos, aplicar la selección de características y entrenar varios modelos. Finalmente, se eligen los modelos más eficaces en función de su F1-score para obtener los resultados más óptimos.

3.1.5. Fase de predicción: detección temprana

Para terminar, una vez hemos entrenado todos los modelos, pasamos a realizar las predicciones para la competición en IberLEF. Como ya he explicado anteriormente, los ficheros que proporciona IberLEF son por rondas, es decir, primero se manda un mensaje y se realiza la predicción sobre ese mensaje. Después, se manda otro mensaje y se predice también, y así en bucle hasta que se mandan todos los mensajes.

	idmessage	nick	round	message	date
0	5086784470	subject184	1	Un poco cansado...	2022-05-23 06:33:17
1	10923780185	subject185	1	No c como decirlo	2020-07-22 22:17:47
2	38934710467	subject186	1	Es la primera vez que...	2020-09-13 05:10:17
3	69981614487	subject188	1	últimamente he pensando...	2022-03-20 20:12:26

Tabla. 3.5: Extracto del fichero de la primera ronda

El tipo de fichero que se iba proporcionando era CSV y tenía el mismo formato que los ficheros de entrenamiento. En la tabla 3.5 tenemos un extracto del fichero de la primera ronda.

Para cada uno se realizaba el mismo proceso de traducción, ingeniería de características que hemos visto. Después, agrupamos los ficheros por id de usuario igual que hemos hecho con los datos de entrenamiento. Finalmente, de todos los modelos se escogen tres para realizar las predicciones y medir métricas de eficiencia:

- RAM usada.
- Porcentaje de uso de CPU.
- Operaciones en coma flotante por segundo (FLOPS).
- El tiempo total del proceso (en milisegundos).
- Kilogramos de emisiones de CO2 (se usa la biblioteca CodeCarbon).

Después de realizar las predicciones y medir las métricas anteriores con cada modelo se suben los resultados al servidor proporcionado por la campaña de evaluación.

En resumen, el objetivo es que nuestros modelos detecten la depresión con un menor número de mensajes, es decir, que la detección de la depresión en un usuario sea cuanto más temprana posible.

3.2. Adaptación al español del sistema de clasificación de depresión

Visto el flujo de trabajo que tenemos en nuestro sistema y descritas todas las partes que lo conforman, vamos a pasar a ver cómo se puede adaptar este para que pueda funcionar con texto en español, es decir, que no sea necesario realizar una traducción previa de los textos a inglés para poder trabajar y simplemente pasemos a la parte de ingeniería de características. Concretamente, esta es la parte que tenemos que modificar. La ingeniería de características que he descrito antes es la que extrae las características, pero está hecha con herramientas para analizar texto en inglés.

El objetivo es sustituir todas esas herramientas que trabajan con texto en inglés por herramientas que cumplan la misma función, pero para texto en español. A conti-

nuación, vamos a ver uno a uno todos los cambios realizados en la fase de ingeniería de características que he realizado.

3.2.1. Términos de la biblioteca Empath

Esta primera parte ha sido difícil de adaptar. La biblioteca empath es una biblioteca que está diseñada principalmente para trabajar con el idioma inglés. Además, en esta parte, cuando llamamos a la función principal de empath para obtener la puntuación de una palabra en un mensaje, lematizamos primero el texto entero. Aquí surge otro problema, ya que el lematizador que usamos es un lematizador de la biblioteca NLTK que se llama WordNetLemmatizer. Este lematizador está creado con el recurso léxico WordNet. WordNet es un diccionario léxico del inglés que agrupa las palabras en conjuntos de sinónimos llamados “synsets” y proporciona relaciones entre estos conjuntos.

La solución a cada uno de estos problemas es la siguiente:

- **biblioteca Empath**: el diccionario que usa esta biblioteca para buscar las palabras que quieres analizar es modificable. Con un script de Python se han extraído todas las palabras de este diccionario, se han traducido al español usando la biblioteca googletrans y después se ha creado otro fichero con el mismo formato y estructura con las palabras en español. Después, se ha modificado el código de la propia biblioteca y en la línea donde se importa el diccionario de palabras en inglés se sustituye por el diccionario en español.
- **Lematizador**: para lematizar los textos, he creado una función auxiliar donde usamos la biblioteca Spacy, para tokenizar los textos en epsañol y extraer el lema de cada token. Después se vuelve a juntar todos los tokens y finalmente se devuelve el texto ya lematizado.

3.2.2. Parte del discurso

La biblioteca Spacy [48] es una de las herramientas más importantes de Procesamiento del Lenguaje Natural. Se utiliza para realizar tareas relacionadas con el procesamiento de texto. Spacy está diseñada para ser eficiente, rápida y fácil de usar. Esta biblioteca se usa para la tokenización, análisis morfológico, lematización, reconocimiento de entidades nombradas (NER), etc.

En nuestro caso, utilizamos esta biblioteca para la Parte del Discurso (Part of Speech). Para usar Spacy, es necesario cargar un modelo pre-entrenado con datos, concretamente nosotros usamos el modelo “en_core_web_lg” que está caracterizado por ser de gran tamaño (“lg” significa “large” o “grande”).

Para poder obtener las etiquetas en español, vamos a sustituir este modelo por uno que también está dentro Spacy que es modelo “es_core_news_lg”. Este modelo, realiza también las mismas funciones que el modelo en inglés, pero para español. También, como el anterior es de gran tamaño.

En esta parte, también contamos los cuantificadores. Esto lo hacemos con una lista hecha a mano de cuantificadores en inglés. Para adaptarlo al español simplemente tenemos que traducir los cuantificadores de la lista y añadir algunos más.

3.2.3. Comentarios tóxicos

En esta parte usamos un modelo de Hugging Face para calcular diferentes grados de toxicidad para cada mensaje. Concretamente, usamos el modelo **unitary/toxic-bert** [49] que solo analiza texto en inglés. Para poder realizar un análisis de toxicidad en los mensajes es necesario cambiar el modelo que estamos usando de Hugging Face. Para ello, hay dos posibles opciones para español que podemos utilizar:

- El modelo **citizenlab/distilbert-base-multilingual-cased-toxicity** [50]: este modelo es un modelo de Hugging Face multilingüe basado en el conjunto de datos “JIGSAW Toxic Comment Classification Challenge”. Este modelo solo nos da puntuación para dos valores diferentes: “not toxic” y “toxic”.
- El modelo **Newtral/xlm-r-finetuned-toxic-political-tweets-es** [51]: este segundo modelo es un modelo basado en el modelo pre-entrenado xlm-roberta-base y esta ajustado sobre un conjunto de datos de tweets de miembros del Congreso de los Diputados Español anotados en función del nivel de toxicidad política que generan. Es un modelo exclusivo del idioma español y es capaz de predecir dos signos de toxicidad política: “toxic” y “very toxic”.

El problema de usar cualquiera de estos dos modelos es que se reducen mucho las características que obtenemos de ellos ya que solo son capaces de dar puntuaciones para dos variables cada uno. En vez de usar uno u otro vamos a usar los dos y a almacenar las características que obtenemos de cada uno.

3.2.4. Análisis de sentimientos

Para realizar el análisis de sentimientos usamos varias herramientas para obtener diferentes características: TextBlob para obtener la subjetividad y la polaridad de los mensajes; y VADER y el modelo cardiffnlp/twitter-roberta-base-sentiment-latest de Hugging Face para obtener valores de los sentimientos del mensaje, es decir, si es positivo, neutral o negativo.

En cuanto a estas herramientas, tenemos algunos problemas. VADER es una herramienta diseñada principalmente para el análisis de sentimientos en inglés. No fue entrenada específicamente para trabajar con otros idiomas. Es posible usarla con textos en español, pero los resultados pueden no ser tan precisos como cuando se utiliza con texto en inglés. Lo mismo ocurre con el modelo de Hugging Face, pero este es sustituible por otro que realice la misma función con texto en español.

Con respecto a este último, lo vamos a sustituir por el modelo **citizenlab/twitter-xlm-roberta-base-sentiment-finetunned** [52], que es un modelo multilingüe que realiza exactamente lo mismo que el otro modelo, nos da una puntuación para tres variables diferentes: positivo, neutral y negativo.

Para sustituir VADER, he usado una biblioteca de Python que se llama **pysentimiento** [53]. pysentimiento es una biblioteca de código abierto que se basa en transformers para realizar tareas relacionadas con el Procesamiento del Lenguaje Natural. Se crea un objeto de esta biblioteca y se usa pasándole como tarea “sentiment”. De esta forma, usa el modelo de Hugging Face que nosotros queremos para que nos de puntuaciones para un texto en tres categorías diferentes: positivo, neutral y negativo.

3.2.5. Negaciones

Para las negaciones, a parte de traducir la lista de las que ya teníamos, he extendido esa lista con otra proporcionada por la tutora de este trabajo, Salud María Jiménez Zafra [54]. Dentro de esta lista, podemos encontrar distintos tipos de negaciones:

- **Negación sintáctica**: una negación sintáctica es aquella en la que se utiliza una negación sintácticamente independiente para expresar la negación. Por ejemplo: no, nunca.
- **Negación léxica**: una negación léxica es aquella palabra cuyo significado tiene un componente negativo. Por ejemplo: negar.

- **Negación morfológica**: este tipo de negación ocurre cuando se utiliza un morfema para expresar la negación. Por ejemplo: ilegal (i-legal).

3.2.6. Legibilidad

En cuanto a la legibilidad vuelve a surgir un problema, ya que, la biblioteca "readibility" es una biblioteca que está diseñada principalmente para textos en inglés y que no es compatible con el idioma español. Sin embargo, existen alternativas y enfoques para calcular la legibilidad de textos en español.

La alternativa que vamos a usar para este procedimiento es **textstat** [55], una biblioteca de Python que es compatible con varios idiomas, incluyendo entre ellos el español. Con esta biblioteca, podemos mantener las mismas variables que obteníamos con readability, por tanto, para esta parte no tenemos ningún problema.

3.2.7. Información de las oraciones

Aquí encontramos el mismo problema que en el apartado anterior, se usa la biblioteca readability para calcular la información sobre las oraciones. En esta parte, calculamos el número de palabras por texto, el número de oraciones por texto, el número de párrafos por texto y tanto el número de palabras largas como de palabras complejas por texto. Para los dos primeros cálculos, vamos a usar también la biblioteca **textstat**, ya que esta nos ofrece funciones para realizar el cálculo automáticamente.

El problema surge a partir del conteo del número de parráfos, ya que textstat no tiene funciones para calcular el resto de operaciones de forma automática. Entonces, para el resto he creado funciones a mano que realizan el cálculo. Para los párrafos realizamos el conteo cada vez que nos encontramos "\n\n" en el texto, porque esto significa un salto de línea.

En cuanto a las palabras largas y cortas, me he basado en la biblioteca readability y en cómo realiza el conteo de estas dos variables. Esta biblioteca considera que una palabra es larga si tiene siete o más caracteres, y una palabra es compleja si tiene más de tres sílabas (también comprobamos si la primera letra es mayúscula para no contar nombres propios). Con esta información, he realizado dos funciones auxiliares para realizar el conteo de cada una de forma manual.

3.2.8. Términos antidepresivos

Simplemente he realizado la traducción de los términos de principios activos de antidepresivos, ya que los nombres propios son iguales en inglés y en español.

3.2.9. Información de las emociones

Aquí también usamos transformers al igual que para comentarios tóxicos o el análisis de sentimientos. En este caso el modelo que usamos de Hugging Face es el **cardiffnlp/roberta-base-emotion** [45] que también está entrenado con un conjunto de datos en inglés y, por tanto, no es recomendable usarlo con texto en español porque los resultados podrían no ser coherentes.

En su lugar, he decidido usar otro modelo que realiza los mismos cálculos que el anterior nombrado y, además, da puntuaciones a emociones que el anterior no tenía. El modelo que he elegido es el modelo **daveni/twitter-xlm-roberta-emotion-es** [56], un modelo entrenado con alrededor de 198 millones de tweets y ajustado para el análisis de emociones en español. Además, como anotación, este modelo fue uno de los modelos propuestos en la campaña de evaluación IberLEF de 2021. Concretamente, se utilizó para una tarea de clasificación de emociones de tweets. Esas emociones eran: enfado (anger), asco (disgust), miedo (fear), felicidad (joy), tristeza (sadness), sorpresa (surprise) y otros (others). Con este modelo consiguieron el primer puesto para esa tarea con una puntuación en la métrica macro-average F1-score de 71’70 %.

3.2.10. Pronombres en primera persona

Como hice con las negaciones, para este paso simplemente es necesario traducir los pronombres en primera persona a español y añadir alguno más.

3.2.11. Términos o expresiones relacionados con la depresión

Finalmente, para terminar de adaptar este script de selección de características, se han traducido todas las expresiones o términos relacionados con la depresión de cada una de las categorías en las que se dividieron. Para realizar la traducción he usado ChatGPT. He pasado cada una de las listas y he unificado también las expresiones que se añadieron posteriormente como lista en el propio script y no en el fichero

de texto. También, añadir que he creado un script que pasa todas las mayúsculas a minúsculas, ya que ChatGPT devolvía muchas expresiones con palabras que empezaban por mayúscula, y para seguir la misma metodología que la selección en inglés las he sustituido por minúsculas.

Para terminar esta sección sobre los pasos realizados en la adaptación al español del script de ingeniería de características comentar aquellas que no han necesitado ningún cambio:

- **Signos de puntuación**: para los signos de puntuación no hay ninguna distinción entre un idioma u otro, esta parte solo realiza el conteo de signos de puntuación que tiene cada mensaje. Por tanto, la forma en la que se había realizado esto funcionaba correctamente para los textos en español.
- **Emoticonos**: de la misma forma que los signos de puntuación, estos son iguales tanto en un idioma como en el otro; y también se realiza un conteo de estos, por lo que no es necesario un cambio en esta parte.
- **Número de letras en mayúscula y número de palabras en mayúscula**: esta parte usa las propias funciones que proporciona Python para ver si una letra o una palabra están en mayúscula, y esto no depende del idioma por lo que se mantiene como está.
- **Información de metadatos (fecha)**: como la columna del DataFrame que contiene este dato es igual en los dos conjuntos de datos (inglés y español), se puede seguir usando esta parte para obtener los intervalos en los que han sido mandados los mensajes y realizar el conteo de cada uno.

Con esto concluye la adaptación al español del script de ingeniería de características. Podemos ver que prácticamente para todas las partes en las que está dividido este script, existen opciones en español que funcionan muy bien y son igual de válidas que las opciones para el inglés.

Pero todavía queda lo más importante, ¿estos cambios afectan al rendimiento de los modelos?, ¿es igual el desempeño del script adaptado al español que el de inglés? En el siguiente capítulo veremos una comparación de los modelos entrenados y de los resultados que obtenemos de cada uno para ver si esta adaptación al español es realmente buena o no vale la pena y es mejor realizar una traducción de los mensajes y usar el sistema que funciona para texto en inglés.

Capítulo 4

RESULTADOS

En este capítulo, vamos a comentar los resultados de los dos sistemas de clasificación. El objetivo es hacerlo de la misma forma en que se utilizó en la tarea MentalRiskES, es decir, realizando una predicción por rondas. Para ello, aunque ya no está disponible el servidor por el que se hacían las peticiones para obtener los mensajes, los organizadores de la tarea MentalRiskES, pertenecientes al grupo de investigación SINAI del departamento de Informática de la Universidad de Jaén, me han proporcionado todos los mensajes de cada ronda para poder realizar esta parte.

Primero realizaré la predicción de los mensajes con el sistema de inglés y se evaluará su rendimiento. Después, haré lo mismo pero con la adaptación que he realizado al español. Finalmente, compararé los dos sistemas y observaremos si la adaptación realizada está a la altura del sistema original, es mejor o es peor.

4.1. Matriz de confusión y métricas a evaluar

Antes de realizar las predicciones, vamos a ver y entender cómo se evalua un sistema de este tipo y las métricas que se usan para ello pero, para entender qué significa cada métrica, primero tenemos que saber qué es una matriz de confusión.

Una matriz de confusión es la herramienta que se utiliza en el campo de la clasificación de problemas para evaluar el rendimiento de un modelo predictivo. Es muy útil cuando se trabaja con problemas de clasificación binaria (como el que enfrentamos en este trabajo). La matriz de confusión organiza la clasificación de los resultados de un modelo en cuatro categorías diferentes:

- **Verdaderos Positivos (VP) o True Positives (TP)**: son instancias que son positivas y el modelo ha clasificado correctamente como positivas.
- **Verdaderos Negativos (VN) o True Negatives (TN)**: son instancias que son negativas y el modelo ha clasificado correctamente como negativas.
- **Falsos Positivos (FP) o False Positives (TP)**: son instancias que son negativas y el modelo ha clasificado incorrectamente como positivas.
- **Falsos Negativos (FN) o False Negatives (FN)**: son instancias que son positivas y el modelo ha clasificado incorrectamente como negativas.

Estos valores se representan en forma de matriz de la siguiente forma:

	Clase Positiva	**Clase Negativa**
Predicción Positiva	VP	FP
Predicción Negativa	FN	VN

Tabla. 4.1: Representación de una matriz de confusión

Usando los valores de una matriz de confusión como la de la Tabla 4.1, se pueden calcular varias métricas de evaluación de rendimiento del modelo. Algunas de estas métricas son las que voy a usar para evaluar el rendimiento de los modelos de este trabajo.

Evaluar un sistema implica medir su rendimientoy precisión, en nuestro caso, en la identificación de señales de depresión en el lenguaje de los usuarios. Para ello, haremos uso de las siguientes métricas:

- **Exactitud (Accuracy)**: la exactitud mide la proporción de predicciones correctas en relación con el total de predicciones. Sin embargo, la exactitud puede ser engañosa si las clases están desbalanceadas (esto quiere decir, que hay muchos más casos de una clase que de otra; por ejemplo, hay mucha más gente que no tiene depresión que gente que sí tiene depresión). Se calcula de la siguiente forma:

$$Exactitud = \frac{NumPrediccionesCorrectas}{NumTotalPredicciones} \tag{4.1}$$

- **Cobertura (Recall)**: la cobertura mide la proporción de instancias de la clase positiva que se identificaron correctamente. Es nuestro caso es útil porque de esta forma no perderíamos casos de depresión. Se calcula de la siguiente manera:

$$Cobertura = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosNegativos} \tag{4.2}$$

- **Especificidad (True Negative Rate)**: la especificidad mide la proporción de instancias de la clase negativa que se identificaron correctamente. Es importante para evitar falsos positivos. Su cálculo se realiza de la siguiente manera:

$$Especificidad = \frac{VerdaderosNegativos}{VerdaderosNegativos + FalsosPositivos} \tag{4.3}$$

- **Precision (Precision)**: la precisión mide la proporción de instancias identificadas como positivas que son realmente positivas. Es útil cuando queremos minimizar los falsos positivos. Para calcularla, tenemos que realizar el siguiente cálculo:

$$Precision = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosPositivos} \tag{4.4}$$

- **F1-Score**: el F1-Score es la media armónica entre la cobertura y la precisión. Es útil cuando se busca un equilibrio entre ambas métricas. Esta media se calcula de la siguiente forma:

$$F1 = 2 * \frac{Precision * Cobertura}{Precision + Cobertura} \tag{4.5}$$

- **Área bajo la curva ROC (AUC-ROC)**: la curva ROC representa la tasa de verdaderos positivos frente a la tasa de falsos positivos en varios umbrales de decisión Un AUC-ROC más alto indica un mejor rendimiento en general.

- **Matthews Correlation Coefficient (MCC)**: es una medida utilizada para evaluar la calidad de la clasificación binaria en problemas de aprendizaje supervisado. Es una métrica que combina la precisión, la cobertura y la especificidad en una sola medida. A diferencia de otras métricas, el MCC tiene en cuenta los cuatro elementos de la matriz de confusión. La fórmula para calcular el MCC es la siguiente:

$$MCC = \frac{VP * VN - FP * FN}{\sqrt{(VP + FP) * (TP + FN) * (VN + FP) * (VN + FN)}} \tag{4.6}$$

A parte de todas estas métricas, también se van a evaluar unas métricas específicas que se calculan gracias al script de evaluación proporcionado por la campaña de evaluación. Gracias a este script, simplemente tenemos que pasar el fichero con los resultados obtenidos por la predicción y este calcula todas las métricas. Estas métricas son las siguientes:

- **ERDE5**: la puntuación de esta métrica indica la probabilidad del modelo de detectar depresión en los 5 individuos que más posibilidades tienen de tener depresión.
- **ERDE50**: esta puntuación es igual que la anterior, pero con los 50 individuos que más posibilidades tienen de tener depresión.
- **latencyTP**: esta métrica indica la media de posts o mensajes que necesita el sistema antes de realizar una predicción en un usuario.
- **latency-weightedF1**: esta medida es el producto del valor F1-Score del modelo con la mediana de un conjunto de sanciones en un rango de [0,1), que se determinan por el tiempo que tarda el modelo en realizar la predicción de cada usuario. Esta sanción es 0 si el número de mensajes necesarios para realizar la predicción es 1. Mientras más mensajes sean necesarios para realizar la predicción, más se acerca a 1 esta sanción.

Una vez explicados todos estos conceptos previos, podemos pasar a ver los resultados de los modelos en el conjunto de datos de entrenamiento para ver qué modelos vamos a usar en las predicciones finales

4.2. Resultados del sistema para texto en inglés

4.2.1. Elección de modelos de clasificación

Antes de poder realizar las predicciones, necesitamos obtener los mejores modelos. Para ello, usaremos la metodología descrita en la Sección 3.1.4 (véase la página 52). La herramienta de AutoML y Pycaret nos permite entrenar diferentes modelos y probarlos con el conjunto de entrenamiento. He elegido modelos de diferentes tipos para entrenar y he obtenido los siguientes resultados:

Como se puede osbervar, la Tabla 4.2 está ordenada por accuracy. De todos los modelos, destacaría los tres primeros:

- **Random Forest Classifier**: este modelo tiene el mayor porcentaje de exactitud, lo que quiere decir que de todos es el que mejor tasa de clasificacicón tiene. Tiene también una alta precisión y una alta cobertura, lo que nos dice que es

eficiente en la identificación de depresión y tiene una tasa baja de falsos positivos. Además, su coeficiente MCC está cerca del 70 %, lo que indica que tiene un buen rendimiento general.

- **Light Gradient Boosting Machine**: este modelo tiene también una exactitud por encima del 80 %, un poco por debajo del anterior, pero sigue siendo un modelo con una buena tasa de clasificación. Su cobertura y precisión también son muy parecidas a las del anterior modelo. Su F1-Score también es alto, lo que nos indica un equilibrio bueno entre precisión y cobertura. En cuanto a su coeficiente MCC, baja un poco con respecto al Random Forest aunque sigue siendo un rendimiento decente.
- **Gradient Boosting Classifier**: este es el peor de los 3 modelos que vamos a usar para los tests por rondas. Su exactitud no llega al 80 %, aunque su F1-Score es prácticamente igual que el Light Gradient Boosting Machine. Su coeficiente MCC cae por debajo del 60 %, indicándonos que no tiene tan buen rendimiento como los dos anteriores, aunque mucho más decente que el resto de modelos.

Modelo	Accuracy	AUC	Recall	Precision	F1-Score	MCC
Random Forest Classifier	0,832	0,910	0,850	0,851	0,844	0,673
Light Gradient Boosting Machine	0,812	0,888	0,825	0,834	0,823	0,633
Gradient Boosting Classifier	0,791	0,876	0,825	0,806	0,811	0,588
Naive Bayes	0,7638	0,845	0,850	0,782	0,802	0,520
Logistic Regression	0,784	0,847	0,800	0,818	0,801	0,576
Decision Tree Classifier	0,729	0,723	0,800	0,750	0,766	0,454
SVM - Linear Kernel	0,710	0,000	0,813	0,735	0,750	0,443
K Neighbors Classifier	0,628	0,688	0,675	0,665	0,664	0,247

Tabla. 4.2: Datos de rendimiento de modelos de predicción para texto en inglés

4.2.2. Análisis de características más importantes

En este caso, al ser el modelo Ranfom Forest el mejor modelo, vamos a analizar la importancia de las características de este modelo.

Como vemos en la Figura 4.1, el modelo le da importancia a muchas características relacionadas con el análisis empático, por ejemplo: nervousness, sadness, suffering,

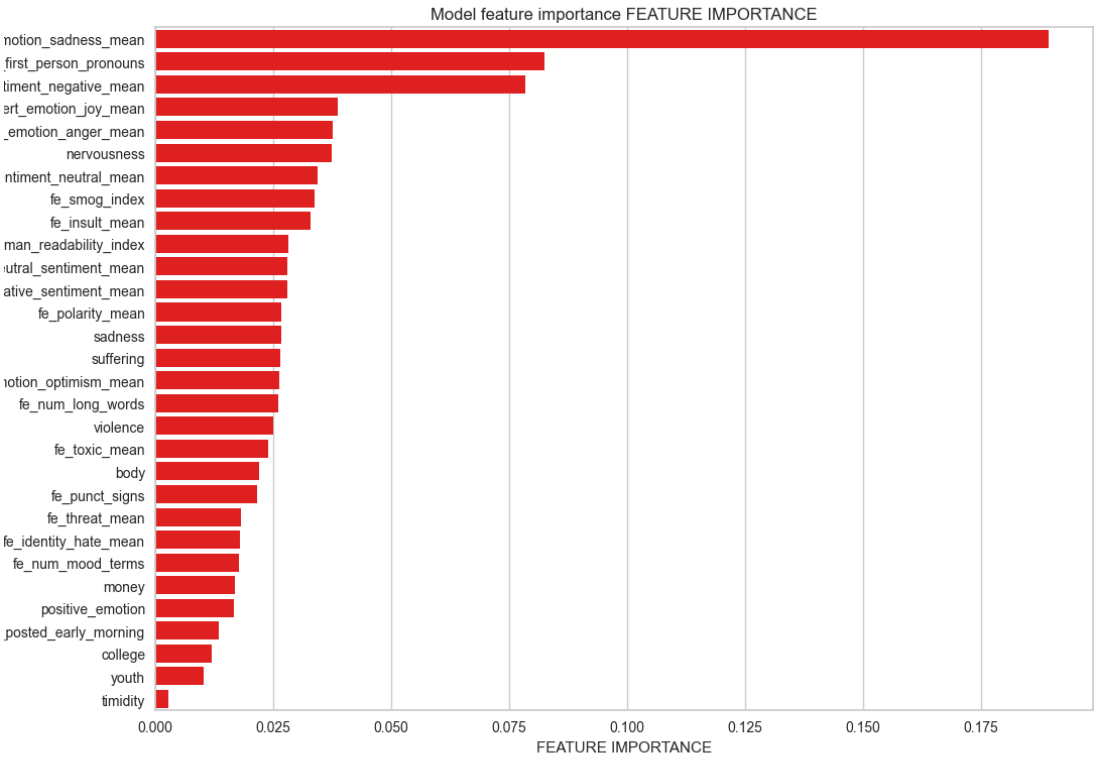

Figura 4.1: Características más importantes para Random Forest.

violence, money, college, youth. Pero se ve claramente como predominan las características extraídas con transformers:

- La media de la puntuación tristeza por usuario: esta características es claramente a la que más importancia le da este modelo. Se obtiene con el transfomer çardiffnlp/roberta-base-emotion":
  - fe_distilbert_emotion_sadness_mean
- Caracterísitcas relacionadas con los sentimientos: tenemos como tercera característica más importante la puntuación media del sentimiento negativo y un poco más abajo la puntuación media del sentimiento neutral. También, están estas dos medidas que se obtienen con VADER como características más importantes:
  - fe_roberta_base_sentiment_negative_mean
  - fe_roberta_base_sentiment_neutral_mean
  - fe_vader_neutral_sentiment_mean
  - fe_vader_negative_sentiment_mean

- También encontramos algunas de las características relacionadas con la toxicidad. Encontramos en la gráfica las puntuaciones medias para el insulto o la toxicidad:
  - fe_insult_mean
  - fe_toxic_mean
- Se observa también que el modelo le da importancia a algunas de las características relacionadas con la legibilidad. Por ejemplo: puntuación del índice SMOG o el índice Coleman-Liau:
  - fe_smog_index
  - fe_coleman_readability_index
- Aunque el modelo no le da mucha importancia a características relacionadas con la Parte del Discurso o información sobre las oraciones, algunas de estas sí que las considera importantes. Encontramos por ejemplo, como segunda características más importante, el número de pronombres en primera persona por mensaje. También, podemos encontrar el número de palabras largas por mensaje o el número de signos de puntuación (fe_num_long_words y fe_punct_signs):
  - fe_num_first_person_pronouns
  - fe_num_long_words
  - fe_punct_signs

Como conclusión, podemos decir que aquellas características relacionadas con los sentimientos, las emociones o la toxicidad, son las que más importancia tienen para identificar signos de depresión en el texto. Además, todas ellas se obtienen con transformers, por lo que se puede decir que juegan un papel importante en la extracción de características y son herramientas muy útiles para problemas de este tipo.

También, es importante la legibilidad del texto y algunas características que nos dan información sobre este, pero el peso que se le da en comparación a las características anteriores es mucho menor, jugando un papel menos importante en el problema.

4.2.3. Resultados obtenidos del conjunto de pruebas

Finalmente, vamos a usar los tres mejores modelos anteriores para obtener los resultados de predicción del conjunto de prueba. Al igual que en la competición, he

ejecutado estos modelos por rondas de forma que he obtenido resultados para cada una de las rondas para cada modelo. Los resultados son acumulativos, esto quiere decir que las predicciones obtenidas para la ronda 3 (por ejemplo) son sobre los mensajes de las rondas 1, 2 y 3.

Para ver los resultados vamos a dividir las métricas en dos partes:

1. En esta primera parte evaluaremos las métricas ERDE5, ERDE50, latencyTP y latency-weightedF1.
2. En la segunda parte evaluaremos el accuracy, macro-F1, macro-recall y macro-precision. Para esta parte además contaremos con los resultados del mejor modelo presentado en la tarea de predicción de MentalRiskES. De esta forma, veremos cuán buenos son los modelos para texto en inglés comparados con el mejor modelo.

Modelo	**ERDE5**	**ERDE50**	**latencyTP**	**latency-weightedF1**
Random Forest Classifier	0,457	0,310	6,0	0,544
Light Gradient Boosting Machine	0,449	0,259	5,0	0,621
Gradient Boosting Classifier	0,402	0,239	4,5	0,638

Tabla. 4.3: Resultados de las métricas ERDE5, ERDE50, latencyTP y latency-weightedF1

Con respecto a la Tabla 4.3, podemos ver que el modelo que mejor rinde es el Random Forest con un 45,7 % de predicciones correctas sobre los 5 usuarios que más probabilidad tienen de tener depresión. Para los 50 usuarios que más probabilidades tienen de tener depresión tiene un porcentaje de 30,1 %. Por otro lado, de los tres modelos, es el modelo que más tarda en predecir a cada usuario con un tiempo de latencia medio de 6 unidades por usuario. Por el contrario, de los tres modelos es el que peor rendimiento medio tiene si observamos su puntuación en latency-weightedF1.

Sin embargo, el modelo LGBM también tiene buenas medidas con respecto a Random Forest, con un 44,8 % en ERDE5, solo un 1 % peor que Random Forest. Donde si baja el porcentaje es en ERDE50, un 5 % más bajo con respecto Random Forest. Su latencia es más baja que la de Random Forest y su rendimiento medio es un poco más alto con una puntuación de 0,62.

Si nos fijamos en la Tabla 4.4, podemos ver que al contrario que en la tabla anterior, el mejor modelo de los tres es el Gradient Boosting. Con una exactitud de 0,6, es el

modelo que más se acerca a la exactitud del mejor modelo presentado en la campaña de evaluación, que tiene una puntuación de 0,71. La precisión y cobertura de este modelo también son ligeramente superiores a las de los otros dos.

Modelo	accuracy	macro-recall	macro-precision	macro-F1
Mejor modelo	0,738	0,756	0,749	0,737
Gradient Boosting Classifier	0,604	0,625	0,617	0,600
Light Gradient Boosting Machine	0,591	0,608	0,602	0,588
Random Forest Classifier	0,584	0,581	0,581	0,581

Tabla. 4.4: Resultados de las métricas accuracy, macro-recall, macro-precision y macro-F1

Aunque el modelo Gradient Boosting es el mejor, la diferencia con respecto a los demás modelos es mínima. Los modelos tiene una puntuación de macro-F1 prácticamente igual, siendo la de Gradien Boosting ligeramente superior. Lo que si podemos decir es que los tres modelos son bastante peores que el mejor modelo, teniendo este unas puntuaciones mucho más altas tanto en exactitud, cobertura, precisión y F1.

Sin embargo, aunque en estas métricas el modelo Gradient Boosting es mejor, en las anteriores es el peor. Esto quiere decir que aunque tenga buena exactitud, no es tan bueno prediciendo esta en los individuos que más probabilidades tienen de tenerlas. Todo lo contrario pasa con el modelo Random Forest. Este es el peor modelo en las métricas correspondientes a la Tabla 4.4, pero al ser la diferencia con respecto a los otros dos modelos ínfima, podemos decir que este modelo podría ser el mejor de los tres. Como se ve en la Tabla 4.3, es el mejor modelo prediciendo depresión en los usuarios que más posibilidades tienen de tenerla, tanto ERDE5 como ERDE50.

4.3. Resultados del sistema para texto en español

En esta sección vamos a realizar el mismo procedimiento que en la sección anterior. Primero obtendremos los mejores modelos de predicción, después analizaremos las características que estos modelos han considerado más importantes para predecir depresión en los usuarios y, finalmente, interpretaremos los resultados obtenidos con estos modelos en la predicción de depresión del conjunto de test.

4.3.1. Elección de modelos de clasificación

Al igual que en la Subsección 4.2.1, lo primero ha sido elegir los mejores modelos de predicción. En este caso la tabla obtenida es la siguiente:

Modelo	Accuracy	AUC	Recall	Precision	F1-Score	MCC
Random Forest Classifier	0,764	0,839	0,800	0,784	0,785	0,532
Gradient Boosting Machine	0,738	0,813	0,738	0,783	0,785	0,493
Light Gradient Boosting Classifier	0,718	0,819	0,763	0,737	0,741	0,441
Naive Bayes	0,677	0,766	0,800	0,676	0,729	0,345
Logistic Regression	0,686	0,727	0,750	0,695	0,714	0,375
Decision Tree Classifier	0,630	0,626	0,650	0,665	0,648	0,262
K Neighbors Classifier	0,496	0,520	0,588	0,538	0,554	-0,035
SVM - Linear Kernel	0,547	0,000	0,513	0,574	0,475	0,133

Tabla. 4.5: Datos de rendimiento de modelos de predicción para texto en español

Al igual que para el sistema de clasifición para texto en inglés, los tres modelos que destacamos en la Tabla 4.5 son los mismos:

- **Random Forest Classifier**: de todos es el modelo que mejores puntuaciones tiene en todas las métricas. Es un modelo con una puntuación en exactitud de 0,76 y una precisión de 0,78. Es el modelo que mejor identifica la depresión como indica su puntuación de 0,8 en cobertura. Su rendimiento general también es el mejor como indica el coeficiente MCC de 0,53.
- **Gradient Boosting Machine**: la diferencia que tenemos en estos resultados con respecto a los resultados de la Subsección 4.2.1 es que el segundo mejor modelo no es el LGBM, si no el modelo Gradient Boosting Machine. No es exactamente mejor pero tiene mejores puntuaciones que LGBM tanto en exactitud, como en precisión, F1-score y coeficiente MCC. Su cobertura es menor que la de LGBM, siendo esta 0,73 mientras que la de LGBM es 0,76. Esto nos dice que aunque es más exacto que el modelo LGBM, es peor identificando los usuarios que sí tiene depresión.
- **Light Gradient Boosting Machine**: como he indicado en la interpretación del modelo anterior, este modelo es el que peores resultados tiene entren los tres modelos que mejores resultados tienen. Este modelo tiene una exactitud superior

a 0,7, y su cobertura es mejor que la del modelo Gradient Boosting Machine. Aunque como ya he indicado, el resto de métricas son un poco peores y su rendimiento general es más bajo que el anterior con un valor de 0,44.

El resto de modelos son simplemente bastante peores que los tres anteriores. Si observamos el rendimiento general en las puntuaciones de los coeficientes MCC, a partir de LGBM, todos caen por debajo de 0,4.

4.3.2. Análisis de características más importantes

En este caso el análisis lo realizaremos también sobre el modelo Random Forest, que es modelo que mejores puntuaciones tiene en las métricas de rendimiento en la Subsección 4.3 anterior.

Como se observa en la Figura 4.2 y como pasaba en la Subsección 4.2.2, las características que más impacto tienen son aquellas relacionadas con el análisis sentimental, el análisis de emociones y la toxicidad. Y a su vez, vuelven a predominar también aquellas características que han sido obtenidas a través de transformers:

- La primera característica y la tercera pertenecen al análisis de emociones: puntuación media de tristeza y puntuación media de otro tipo de emociones. También podemos encontrar un poco más abajo la puntuación media del miedo o la puntuación media de asco:
  - fe_daveni_emotion_sadness_mean
  - fe_daveni_emotion_others_mean
  - fe_daveni_emotion_fear_mean
  - fe_daveni_emotion_disgust_mean
- Destacan las características relacionadas con el análisis de sentimientos y polaridad. Podemos ver que la segunda característica que más impacto tiene es la puntuación media de negatividad que obtenemos gracias a la librería pysentimiento (que como mencioné en el apartado 3.2.4, es una librería basada en transformers). También, entre las características que más impacto tienen, encontramos la puntuación media de sentimiento negativo (obtenida con el transfomer çitizenlab/twitter-xlm-roberta-base-sentiment-finetunned") y la puntuación media de sentimiento positivo con pysentimiento:

  - fe_pysentimiento_negative_mean
  - fe_roberta_base_negative_mean
  - fe_pysentimiento_positive_mean

- También se considerna características con impacto algunos de las métricas de legibilidad, como el índice lix, el índice Coleman-Liau o el índice Dale-Chall:
  - fe_lix_textstat_index
  - fe_coleman_textstat_index
  - fe_dale_chall_textstat_index

- Con menos impacto podemos encontrar también características relacionadas con información sobre el texto, o la hora de publicación de los mensajes.

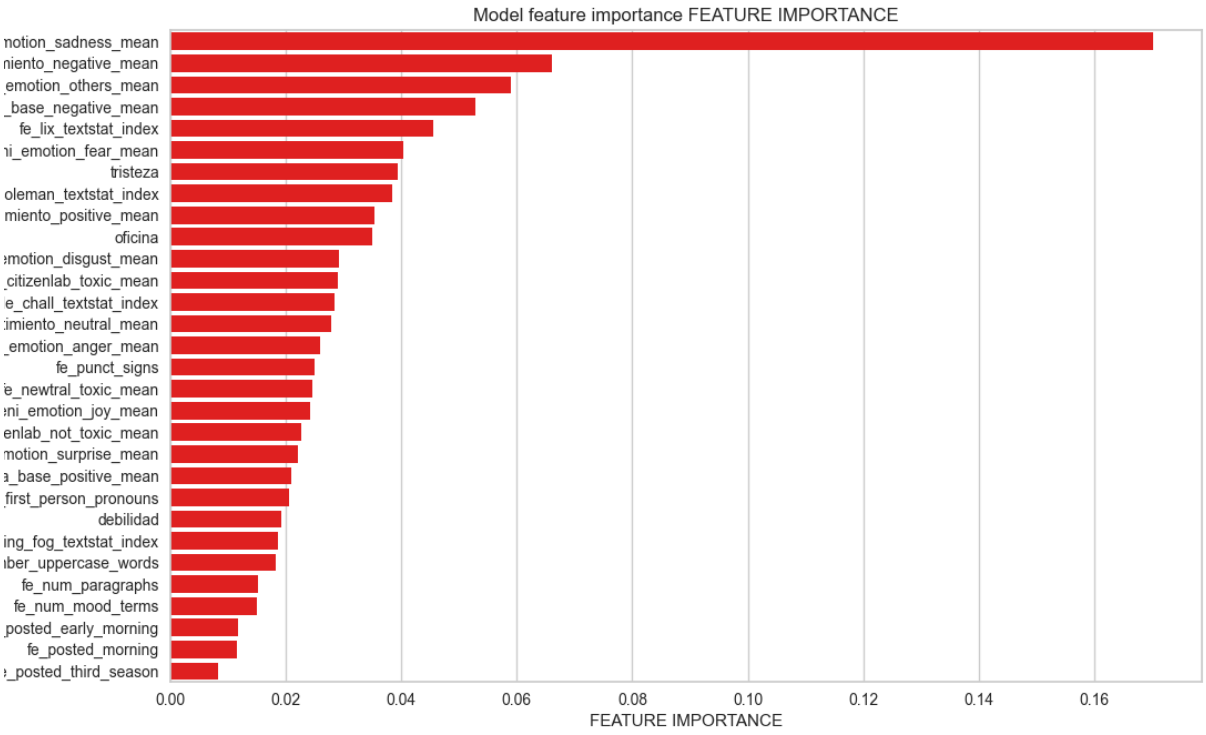

Figura 4.2: Características más importantes para Random Forest para texto en español.

Podemos concluir, al igual que ocurría con el sistema de clasificación para texto en inglés, que, al final, las características más importantes para detectar la depresión son aquellas relacionadas con el análisis de sentimientos, análisis de emociones o análisis de toxicidad. Estas características son aquellas que más información pueden proporcionar en cuanto a si una persona puede padecer o no depresión.

4.3.3. Resultados obtenidos en el conjunto de pruebas

Por último, vamos a ver los resultados de esta adaptación del sistema de clasificación antes de compararlos con los resultados del sistema anterior, para ver las diferencias entre los dos sistemas.

Modelo	ERDE5	ERDE50	latencyTP	latency-weightedF1
Gradient Boosting Classifier	0,300	0,219	3,0	0,657
Light Gradient Boosting Machine	0,285	0,214	3,0	0,670
Random Forest Classifier	0,261	0,201	3,0	0,684

Tabla. 4.6: Resultados de las métricas ERDE5, ERDE50, latencyTP y latency-weightedF1

Al realizar la adaptación y obtener los resultados, como se puede ver en la Tabla 4.6, el mejor de los tres modelos para estas métricas es el Gradient Boosting (al contrario que antes, que el mejor era Random Forest).

Gradient Boosting tiene un 30 % de predicciones correctas para ERDE5, pero baja bastante en ERDE50, con un 21,9 % de predicciones correctas. LGBM no dista mucho de estos resultados, siendo un 2 % más bajo en predicciones correctas en ERDE5, y teniendo prácticamente el mismo reultado para ERDE50. El peor de los tres es Random Forest, donde obtenemos un 26 % para ERDE5 y un 20 % para ERDE50. Estos resultados nos dicen que estos modelos entrenados con la adaptación al español, no son muy buenos detectanto depresión, ya que sus porcentajes de predicciones correctas para aquellos usuarios que tienen más posibilidad de tener depresión son bastante bajos.

En cuanto a la velocidad, los tres modelos tienen una media de 3 unidades (mensajes) por predicción, es decir, la media de posts necesarios para medir la depresión son 3. En cuanto al rendimiento general, el mejor modelo es Random Forest con un 0,68 de puntuación en latency-weightedF1. De cerca, le sigue LGBM con un 0,67 y el peor (teniendo en cuenta que la diferencia es muy baja entre los tres) es Gradient Boosting, que tiene una puntuación de 0,66.

Con respecto a las métricas de rendimiento, si observamos la Tabla 4.7, observamos que Random Forest es el modelo que mejor resultados obtiene. presenta un nivel de precisión macro de 0,591, lo que indica que, en promedio, el 59,1 % de las predicciones son correctas. Tiene una puntuación de 0,59 en accuracy, lo que nos indica que no es un modelo muy bueno realizando predicciones. Aun así, su cobertura es

de 0,72, lo que nos da a entender que es bueno prediciendo los casos positivos de depresión. Incluso se acerca bastante a la cobertura del mejor modelo presentado en la tarea. Sus valores de precision y F1, 0,621 y 0,551 respectivamente, indican que existe un equilibrio entre la capacidad de identificar casos positivos y la minimización de falsos positivos.

El modelo LGBM no dista mucho de Random Forest, teniendo una diferencia de puntuaciones de un 2 % o 3 % más bajas. Destaca menos en cobertura, cayendo esta por debajo del 70 %. Por otro lado, el peor de los tres modelos es el Gradient Boosting. Su exactitud está por debajo del 55 %, es decir, el modelo solo predice correctamente un 55 % de los casos. Su cobertura es más alta, un 68 % de las veces clasifica correctamente los casos positivos de depresión, pero su puntuación de macro-F1 de 0,47 nos da a entender que su rendimiento es moderado en términos de equilibro de precisión y cobertura.

Modelo	Accuracy	macro-recall	macro-precision	macro-F1
Mejor modelo	0,738	0,756	0,749	0,737
Random Forest Classifier	0,591	0,722	0,621	0,551
Light Gradient Boosting Machine	0,570	0,693	0,601	0,527
Gradient Boosting Classifier	0,537	0,686	0,572	0,473

Tabla. 4.7: Resultados de las métricas accuracy, macro-recall, macro-precision y macro-F1

En resumen, los resultados proporcionan una imagen clara de las fortalezas y debilidades de cada modelo. El Random Forest Classifier destaca por su alto recall, el Light Gradient Boosting Machine por un buen equilibrio entre precisión y recall, mientras que el Gradient Boosting Classifier puede requerir ajustes para mejorar su rendimiento general.

4.4. Comparativa entre los modelos para texto en inglés y los modelos para texto en español

Por último, y antes de sacar las conclusiones finales de este trabajo, vamos a comparar los modelos que trabajan con texto en inglés y los modelos que trabajan con texto en español para ver si la adaptación realizada está o no a la altura del sistema original.

Modelo	accuracy	macro-recall	macro-precision	macro-F1
Gradient Boosting Classifier-EN	0,604	0,625	0,617	0,600
Random Forest Classifier-ES	0,591	0,722	0,621	0,551
Light Gradient Boosting Machine-EN	0,591	0,608	0,602	0,588
Random Forest Classifier-EN	0,584	0,581	0,581	0,581
Light Gradient Boosting Machine-ES	0,570	0,693	0,601	0,527
Gradient Boosting Classifier-ES	0,537	0,686	0,572	0,473

Tabla. 4.8: Comparativa entre modelos de predicción para texto en inglés y español

La Tabla 4.8 está ordenada de más a menos exactitud, por tanto, fijándonos únicamente en esta métrica, podríamos decir que, quitando el modelo Random Forest, los modelos para clasificar texto en español son peores. Pero, al mirar los resultados para la cobertura, podemos ver que los tres modelos que clasifican texto en español son superiores a los otros tres modelos, indicándonos que los modelos para texto en español tienen un porcentaje más alto de predicciones correctas para casos positivos de depresión. Este dato es muy importante, ya que, el objetivo final de estos modelos sería ayudar a profesionales a detectar depresión en pacientes, por lo que estos modelos serían mejores para esta tarea.

La precisión varía entre los modelos, sin una tendencia clara basada en el idioma. Sin embargo, los modelos en español (Random Forest Classifier-ES y Light Gradient Boosting Machine-ES) tienden a tener valores más altos.

Finalmente, la puntuación F1 macro, que equilibra precisión y cobertura, muestra una pequeña superioridad para el idioma inglés. Por tanto, en cuanto a rendimiento general, se podría decir que son ligeramente superiores a los modelos para idioma en español.

Capítulo 5

CONCLUSIONES

5.1. Resultados obtenidos

Durante este trabajo se han realizado y completado todos los objetivos propuestos inicialmente para este proyecto. Se ha realizado una investigación sobre el estado del arte del Procesamiento del Lenguaje Natural para poner en contexto este trabajo y ver cómo se puede aplicar esta rama de la Inteligencia Artificial, centrándonos finalmente en las tareas de clasificación como la que se ha llevado a cabo. También, se ha descrito y explicado qué son y cómo funciona las campañas de evaluación. Gracias a esto se ha podido ver y aplicar cómo se evaluan estos sistemas de clasificación dependiendo del tipo que sean.

Como se sabe, este trabajo es trabajo es teórico-experimental, ya que no se ha tenido que desarrollar un sistema de clasificación desde cero. La parte experimental de este trabajo, ha sido la modificación y adaptación del sistema existente para que pueda analizar y clasificar texto en español. Esta parte ha sido llevada a cabo con éxito, realizando todos los cambios requeridos para el correcto funcionamiento del sistema.

Por último, una vez analizado el sistema inicial y, posteriormente, adaptado al español; se ha llevado a cabo la fase de experimentación donde hemos obtenido los resultados sobre los conjuntos de datos proporcionados por la campaña de evaluación. Se ha evaluado cómo funciona cada uno de los sistemas y se han comparado para ver si la modificación realizada es una buena adaptación.

5.2. Conclusiones sobre los resultados finales

Finalmente, hemos llegado a la respuesta para la pregunta qué nos hemos hecho a lo largo de este trabajo, ¿es una buena adaptación del sistema de clasificación inicial? Pues depende de para qué se quieran usar estos modelos. Si el objetivo es clasificar a todas las personas y saber si tienen o no depresión de forma directa, podemos decir que los 6 modelos finales son relativamente parecidos. Si nuestro objetivo es determinar exclusivamente si la persona tiene depresión, podemos decir que los tres modelos que clasifican texto en español son mejores para esta tarea. En cuanto a valores generales los seis modelos son parecidos. El objetivo que nos propongamos es lo que determinará el uso de uno u otro.

5.3. Posibles mejoras

Uno de los problemas principales que he tenido a la hora de realizar este trabajo, ha sido encontrar herramientas equivalentes a las que se usan para elegir las características que queremos examinar de los textos. Considero que hay muchas menos alternativas para analizar el texto en español que en inglés.

Una gran parte de los cambios ha sido usar muchos transformers para algunas tareas que no los requerían en el análisis de texto en inglés. Esto ralentiza, de gran manera, la ejecución del script de ingeniería de características. Por suerte, los conjuntos de datos no han sido relativamente extensos. Hablamos de unos 7000 mensajes aproximadamente. En otro caso, utilizar un gran cantidad de transfomers puede ser imposible si nuestro conjunto de datos es mucho más extenso.

Otra de las partes que ha ralentizado el proceso que he comentado, es la del análisis empático. No llegué a encontrar una herramienta equivalente a esta en español, y el realizar la lematización de las palabras de cada mensaje con una función auxiliar hecha a mano, hace que este análisis sea mucho más lento. Por lo que una posible mejora, sería utilizar una biblioteca que fuese equivalente o realizar una función auxiliar más eficiente. Quizá, con una búsqueda más extensa de posibles herramientas, encontraría una herramienta que hiciese el mismo trabajo.

5.4. Satisfacción personal

El trabajo que he realizado ha sido satisfactorio. El trabajo me ha llevado más tiempo del que esperaba, pero considero que los resultados finales y la adaptación realizada me han permitido obtener un gran conocimiento sobre el Procesamiento el Lenguaje Natural y el aprendizaje automático.

Considero que ha sido satisfactorio porque he llevado a cabo todos los objetivos que fijé al inicio del trabajo, con unos resultados finales bastante buenos. Si nos fijamos en los resultados obtenidos por los modelos para texto en español, son bastante parecidos a los modelos para texto en inglés; incluso mejores en algunas métricas. En cuanto a la parte de programación, no ha sido tarea fácil adaptar el sistema. Aprender cómo funciona un código con el que no estás familiarizado y no has visto nunca no es una tarea sencilla, primero tuve que aprender cómo funcionaba el sistema para posteriormente poder modificarlo.

Finalmente, decir que acabo contento con la modificación realizada y con los resultados obtenidos, ya que desde un primer momento no sabía si los cambios iban a acabar dando buenos resultados, y creo que al final han sido unos resultados bastante decentes.

Bibliografía

[1] Ginebra Organización Mundial de la Salud. Informe mundial sobre salud mental: transformar la salud mental para todos. panorama general. *Organización Mundial de la Salud*, 2022.

[2] Mármol-Romero A.M Moreno-Muñoz-A. Plaza-del-Arco F.M. Molina-González M.D. Martín-Valdivia M.T. Ureña-López, A. Overview of MentalRiskES at IberLEF 2023: Early Detection of Mental Disorders Risk in Spanish. Procesamiento del Lenguaje Natural. 71:329–250, 2023.

[3] Jiménez Zafra-S.M. Rangel F. Gómez, M.M.Y. Overview of IberLEF 2023: Natural Language Processing Challenges for Spanish and other Iberian Languages. *In Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2023), co-located with the 39th Conference of the Spanish Society for Natural Language Processing (SEPLN 2023)*, 2023.

[4] De Choudhury-M. Gamon M.-Counts-S. Horvitz, E. Predicting Depression via Social Media. *Proceedings of the International AAAI Conference on Web and Social Media*, 7:128–137, 2021. doi: https://doi.org/10.1609/icwsm.v7i1.14432.

[5] Página oficial de la ONG SoGooData. URL https://sogooddata.org/index.php/colabora/. comprobado en 2024-02-19.

[6] Fernández-Hernández A.-Moreno-Sánchez R.-Viosca-Ros J. Enrique-Guillén R. Cruz-Dıaz N. P. Jiménez-Zafra, S. M. TextualTherapists at MentalRiskES-IberLEF2023: Early Detection of Depression using a Userlevel Feature-based Machine Learning Approach. *In IberLef (Working Notes). CEUR Workshop Proceedings*, 2023.

[7] A Turing. Computing machinery and intelligence. *Theories of Mind: An Introductory Reader*, page 51, 2006.

[8] Elizabeth D Liddy. Natural language processing. 2001.

[9] Dorys Moreira, Ivan Cruz, Karolina Gonzalez, Andrea Quirumbay, Christian Magallan, Teresa Guarda, Alicia Andrade, and Carlos Castillo. Análisis del estado actual de procesamiento de lenguaje natural. *Revista Ibérica de Sistemas e Tecnologias de Informação*, (E42):126–136, 2021.

[10] Hyesil Jung, Hyeoun-Ae Park, and Tae-Min Song. Ontology-based approach to social data sentiment analysis: detection of adolescent depression signals. *Journal of medical internet research*, 19(7):e259, 2017.

[11] Nam Hyeok Kim, Ji Min Kim, Da Mi Park, Su Ryeon Ji, and Jong Woo Kim. Analysis of depression in social media texts through the patient health questionnaire-9 and natural language processing. *Digital Health*, 8:20552076221114204, 2022.

[12] Página de inicio la última edición celebrada del workshop TREC. URL https://trec.nist.gov/celebration/25thcelebration.html. comprobado en 2024-02-20.

[13] Página de inicio de la última edición celebrada del workshop WMT. URL https://www2.statmt.org/wmt24/index.html. comprobado en 2024-02-20.

[14] Página de inicio de la última edición celebrada del workshop SemEval. URL https://semeval.github.io/SemEval2023/. comprobado en 2024-02-20.

[15] Página de inicio del workshop DSCT. URL https://dstc10.dstc.community/home. comprobado en 2024-02-20.

[16] Página de inicio de la última edición celebrada del workshop EVALITA. URL https://www.evalita.it/campaigns/evalita-2023/. comprobado en 2024-02-20.

[17] Página de inicio de la última edición celebrada del workshop CLEF. URL https://clef2023.clef-initiative.eu/. comprobado en 2024-02-20.

[18] Página de inicio de la última edición celebrada del workshop TPDL. URL https://tpdl2023.dei.unipd.it/index.html. comprobado en 2024-02-20.

[19] Página de inicio de la última edición celebrada del workshop TASS. URL http://tass.sepln.org/2020/. comprobado en 2024-02-20.

[20] Página de la última edición celebrada del workshop IberEval, . URL https://sites.google.com/view/ibereval-2018/home?authuser=0. comprobado en 2024-02-20.

[21] Página de la última edición celebrada del workshop IberLEF, . URL https://sites.google.com/view/iberlef-2023/home. comprobado en 2024-02-20.

[22] Página de la última edición de MentalRiskES. URL https://sites.google.com/view/mentalriskes. comprobado en 2024-02-20.

[23] Emoji Sentiment Ranking, . URL https://kt.ijs.si/data/Emoji_sentiment_ranking/. comprobado en 2024-02-20.

[24] Documentación de la biblioteca googletrans de Python. URL https://pypi.org/project/googletrans/. comprobado en 2024-02-20.

[25] Documentación de los modelos de OpenAI. URL https://platform.openai.com/docs/models/gpt-base. comprobado en 2024-02-20.

[26] Documentación del modelo basado en transformers de HuggingFace Helsinki-NLP/opus-mt-es-en. URL https://huggingface.co/Helsinki-NLP/opus-mt-es-en. comprobado en 2024-02-20.

[27] Documentación de la API de DeepL. URL https://www.deepl.com/docs-api. comprobado en 2024-02-20.

[28] Documentación de la biblioteca Pycaret de Python. URL https://pycaret.org/. comprobado en 2024-02-20.

[29] Documentación de la biblioteca Empath de Python. URL https://github.com/Ejhfast/empath-client. comprobado en 2024-02-20.

[30] Documentación de la biblioteca oficial de VADER para Python. URL https://pypi.org/project/vaderSentiment/. comprobado en 2024-02-20.

[31] Documentación de la biblioteca oficial de TextBlob para Python, . URL https://textblob.readthedocs.io/en/dev/. comprobado en 2024-02-20.

[32] Documentación del modelo basado en transformers de HuggingFace cardiffnlp/twitter-roberta-base-sentiment. URL https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment. comprobado en 2024-02-20.

[33] Documentación de la biblioteca transformers de Python. URL https://pypi.org/project/transformers/. comprobado en 2024-02-20.

[34] Documentación de la biblioteca pipeline de Python. URL https://pypi.org/project/pipeline/. comprobado en 2024-02-20.

[35] Documentación de la versión 1.0 de la biblioteca emot de Python, . URL https://pypi.org/project/emot/1.0/. comprobado en 2024-02-20.

[36] Documentación de la biblioteca readability de Python. URL https://pypi.org/project/readability/. comprobado en 2024-02-20.

[37] Marina Solnyshkina, Radif Zamaletdinov, Ludmila Gorodetskaya, and Azat Gabitov. Evaluating text complexity and Flesch-Kincaid grade level. *Journal of social studies education research*, 8(3):238–248, 2017.

[38] Edgar A Smith and RJ Senter. *Automated readability index*, volume 66. Aerospace Medical Research Laboratories, Aerospace Medical Division, Air . . . , 1967.

[39] Akhil Kher, Sandra Johnson, Robert Griffith, et al. Readability assessment of online patient education material on congestive heart failure. *Advances in preventive medicine*, 2017, 2017.

[40] George W England, Margaret Thomas, and Donald G Paterson. Reliability of the original and the simplified Flesch reading ease formulas. *Journal of Applied Psychology*, 37(2):111, 1953.

[41] Damian Świeczkowski and Sławomir Kułacz. The use of the Gunning Fog Index to evaluate the readability of Polish and English drug leaflets in the context of Health Literacy challenges in Medical Linguistics: An exploratory study. *Cardiology Journal*, 28(4):627–631, 2021.

[42] Jonathan Anderson. Lix and rix: Variations on a little-known readability index. *Journal of Reading*, 26(6):490–496, 1983.

[43] Shixiang Zhou, Heejin Jeong, and Paul A Green. How consistent are the best-known readability equations in estimating the readability of design standards? *IEEE Transactions on Professional Communication*, 60(1):97–111, 2017.

[44] Wayne D Lee and Bernard R Belden. A cross-validation readability study of general psychology textbook material and the Dale-Chall Readability Formula. *The Journal of Educational Research*, 59(8):369–373, 1966.

[45] Documentación del modelo basado en transformers de HuggingFace cardiffnlp/roberta-base-emotion. URL https://huggingface.co/cardiffnlp/roberta-base-emotion. comprobado en 2024-02-20.

[46] Documentación de la biblioteca NRCLex de Python. URL https://pypi.org/project/NRCLex/. comprobado en 2024-02-20.

[47] Kurt Kroenke, Robert L Spitzer, and Janet BW Williams. The PHQ-9: validity of a brief depression severity measure. *Journal of general internal medicine*, 16(9): 606–613, 2001.

[48] Documentación de la biblioteca Spacy para Python. URL https://spacy.io/usage. comprobado en 2024-02-20.

[49] Documentación del modelo basado en transformers de HuggingFace unitary/toxic-bert. URL https://huggingface.co/unitary/toxic-bert. comprobado en 2024-02-20.

[50] Documentación del modelo basado en transformers de HuggingFace citizenlab/distilbert-base-multilingual-cased-toxicity, . URL https://huggingface.co/citizenlab/distilbert-base-multilingual-cased-toxicity. comprobado en 2024-02-20.

[51] Documentación del modelo basado en transformers de HuggingFace Newtral/xlm-r-finetuned-toxic-political-tweets-es. URL https://huggingface.co/Newtral/xlm-r-finetuned-toxic-political-tweets-es. comprobado en 2024-02-20.

[52] Documentación del modelo basado en transformers de HuggingFace citizenlab/twitter-xlm-roberta-base-sentiment-finetunned, . URL https://huggingface.co/citizenlab/twitter-xlm-roberta-base-sentiment-finetunned. comprobado en 2024-02-20.

[53] Documentación de la biblioteca pysentimiento de Python. URL https://github.com/pysentimiento/pysentimiento. comprobado en 2024-02-20.

[54] S.M Jiménez Zafra. Negation processing in spanish and its application to sentiment analysis. *Doctoral dissertation, Universidad de Jaén*, 2019. doi: https://hdl.handle.net/10953/1108.

[55] Documentación de la biblioteca textstat de Python, . URL https://textstat.readthedocs.io/en/latest/. comprobado en 2024-02-20.

[56] Documentación del modelo basado en transformers de HuggingFace daveni/twitter-xlm-roberta-emotion-es. URL https://huggingface.co/daveni/twitter-xlm-roberta-emotion-es. comprobado en 2024-02-20.