

## Facultat Internacional de Comerç i Economia Digital La Salle

Trabajo Final de Máster

Máster Universitario en Ciencia de los Datos / Data Science

**Estudio del odio en las redes sociales  
mediante técnicas de NLP  
en el proyecto Disargue**

Alumno

David Larrosa Camps

Profesor Ponente

Dr. Xavier Vilasís i Cardona

---

# **ACTA DEL EXAMEN**

## **DEL TRABAJO FINAL DE MÁSTER**

---

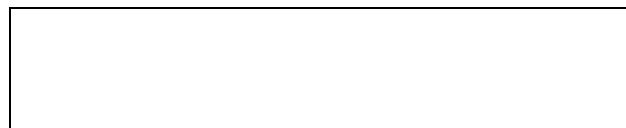
Reunido el Tribunal calificador en el día de la fecha, el alumno

**David Larrosa Camps**

Expuso su Trabajo de Final de Máster, el cual trató sobre el tema siguiente:

Estudio del odio en las redes sociales mediante técnicas de NLP en el proyecto  
Disargue

Acabada la exposición y contestadas por parte del alumno las objeciones formuladas por los miembros del tribunal, este valoró el mencionado Trabajo con la calificación de



Barcelona,

VOCAL DEL TRIBUNAL

VOCAL DEL TRIBUNAL

PRESIDENTE DEL TRIBUNAL



## Resumen

El uso de las redes sociales se ha convertido en una actividad habitual para la mayoría de las personas. Aunque tengan muchas ventajas, tienen el inconveniente de permitir ciertos comportamientos tóxicos que pueden ser dañinos para algunas personas. La moderación de dichos comportamientos ha sido la solución que actualmente se ha estado aplicando en las redes sociales. No obstante, no ha sido un método especialmente exitoso debido a que el proceso actual de moderación es en parte manual, lento y no puede seguir el ritmo a la creación de contenido, por lo que se abre la necesidad de crear mecanismos automáticos para la detección del odio.

Este proyecto tiene como objetivo dos puntos principales: primero, analizar estos comportamientos tóxicos para entender qué son y qué los caracteriza, y el segundo objetivo consiste en crear una herramienta para poder detectarlos de manera automática.

En este documento se puede ver cómo primero se estudia qué es el odio y el estado del arte de su detección, luego, se continúa con una parte de análisis de los datos, donde se empieza estudiando qué es el Procesamiento del Lenguaje Natural (NLP) y sus conceptos básicos, y por último se analizan los datos que se usarán durante el proyecto. A continuación, se realizará la parte más técnica, donde se aplicarán algoritmos de Machine Learning (ML), Deep Learning (DL) y finalmente modelos de lenguaje (LLM).

En la parte de ML se verán los modelos de *Support Vector Machine* (SVM), *Random Forest* y XGBoost, luego, en el apartado de DL, se verán distintas arquitecturas de redes neuronales como *Multi Layer Perceptrons* (MLP), *Recurrent Neural Networks* (RNN), combinaciones de MLP con *fine tuning* de modelos BERT, combinaciones de MLP con RNN, y finalmente MLP con RNN con *fine tuning* de modelos BERT. En el apartado de LLM se usan modelos especializados en la detección de odio y modelos generales para poder clasificar y así usar dichas respuestas para comparar si el etiquetaje con el original.

El resultado de este proyecto es un modelo que consigue clasificar un texto en función de si contiene odio o no, y un análisis de los distintos componentes del odio en el contexto de las elecciones europeas del año 2024.

Este proyecto se llama *Disargue* y se ha realizado en colaboración con la Universidad de Blanquerna.

## Resum

L'ús de les xarxes socials s'ha convertit en una activitat habitual per a la majoria de les persones. Tot i que aporten molts avantatges, tenen l'inconvenient de permetre certs comportaments tòxics que poden ser perjudicials per a algunes personnes. La moderació d'aquests comportaments ha sigut la solució que actualment s'ha estat aplicant a les xarxes socials. No obstant, no ha sigut un mètode especialment exitós degut a que el procés actual de moderació és en part manual, lent, i no pot seguir el ritme de creació de contingut. Tot això ha generat la necessitat de crear mètodes automàtics per a la detecció d'odi.

Aquest projecte té com a objectiu dos punts principals: primer, analitzar els comportaments tòxics per entendre què són y què els caracteritza, y el segon objectiu consisteix en crear una eina per poder detectar-los de manera automàtica.

En aquest document es pot veure com primer s'ha estudiat què és l'odi i l'estat de l'art en la seva detecció. Després, es continua amb una part d'anàlisi de les dades, on es comença estudiant què és el Processament del Llenguatge Natural (NLP) i els seus conceptes bàsics, i per últim, s'analitzen les dades que s'usaran durant el projecte. A continuació, es realitzarà la part més tècnica, on s'aplicaran mètodes de *Machine Learning* (ML), *Deep Learning* (DL) i finalment models de llenguatge (LLM).

En la part d'ML es veuran els models de *Support Vector Machine* (SVM), *Random Forest* i XGBoost. Després, en l'apartat de DL es veuran varíes arquitectures de xarxes neuronals com ara *Multi Layer Perceptrons* (MLP), *Recurrent Neural Networks* (RNN), combinacions de MLP amb *fine tuning* de models BERT, combinacions de MLP amb RNN, i finalment, MLP amb RNN amb *fine tuning* a models BERT. En l'apartat de LLM es faran servir tant models especialitzats en la detecció d'odi com models generals per poder realitzar una classificació de les dades i així poder fer servir les respostes per comparar-les amb l'etiquetatge original.

El resultat d'aquest projecte és un model que aconsegueix classificar un text en funció de si conté odi o no, i un anàlisis dels diferents components de l'odi en el context de les eleccions europees de l'any 2024.

Aquest projecte es diu *Disargue* i s'ha realitzar en col·laboració amb la Universitat Blanquerna.

## Abstract

The use of social networks has become a common activity for most people. Although they offer many advantages, they also have the downside of enabling certain toxic behaviours that can be harmful to some individuals. Moderation of these behaviours has been the current solution applied on social networks. However, it has not been a particularly successful method because the current moderation process is partially manual, slow, and unable to keep up with the pace of content creation. All this has created the need to develop automatic methods for hate detection.

This project has two main objectives: first, to analyze toxic behaviours to understand what they are and what characterizes them. The second goal is to develop a tool capable of detecting them automatically.

This document shows how the study began with an analysis of what hate is and the state of the art in its detection. Then, it continues with a data analysis section, beginning with an introduction to what Natural Language Processing (NLP) is and its basic concepts, followed by an analysis of the data that will be used throughout the project. Next comes the more technical part, where Machine Learning (ML), Deep Learning (DL), and finally Large Language Models (LLM) methods are applied.

In the ML section, models such as Support Vector Machine (SVM), Random Forest, and XGBoost are used to classify the data. Then, in the DL section, various neural network architectures are presented, such as Multi-Layer Perceptrons (MLP), Recurrent Neural Networks (RNN), combinations of MLP with fine tuned BERT models, combinations of MLP with RNN, and finally, MLP with RNN with fine tuned BERT models. In the LLM section, both hate detection specific models and general-purpose models are used to classify the data and compare their outputs with the original labels.

The result of this project is a model capable of classifying text based on whether it contains hate or not, and an analysis of the different components of hate in the context of the 2024 European elections.

This project is called *Disargue* and was carried out in collaboration with Blanquerna University.

## **Agradecimientos**

Considero que este proyecto ha sido posible gracias a todas las personas que me rodean y que me han estado acompañando durante todo este proceso. Me gustaría agradecer a Anna el amor incondicional y el soporte que ha hecho durante todo el máster. También me gustaría agradecer a mis padres, a mi hermana y a mi abuela haber estado siempre a mi lado animando para que consiga mis objetivos.

También me gustaría agradecer a Xavier Vilasís la tutela y los conocimientos que me ha estado enseñando, no solo durante este proyecto, sino también durante la carrera y el máster.

Finalmente, también me gustaría agradecer su compañía a los compañeros del MUDS. Ha sido un curso duro donde todos hemos crecido tanto personal como profesionalmente. Me alegra mucho haber tenido un grupo así de bueno.

Gracias a todos por vuestros ánimos y soporte.

# Índice

1	Introducción.....	10
2	Objetivo principal .....	11
3	Marco teórico .....	12
3.1	Definición de odio .....	12
3.2	Implicaciones éticas y sociales de los discursos de odio .....	14
4	Procesamiento del Lenguaje Natural (NLP) .....	15
4.1	Conceptos NLP y Ciencia de Datos .....	16
4.2	Problemas con NLP para la detección de odio .....	19
4.3	Creación de un dataset para el estudio del odio.....	20
4.4	Métodos para la detección del odio con NLP .....	21
5	Estado del arte .....	22
5.1	Resumen estudios previos.....	24
6	Datos .....	26
6.1	Datasets .....	26
6.2	EDA .....	27
6.2.1	Estudio de los textos.....	27
6.2.2	Estudio de los emojis.....	41
7	Dataset .....	50
7.1	Script Youtube.....	50
7.2	Transformaciones de textos / limpieza.....	52
7.3	Encodings y embeddings .....	53
7.3.1	One-hot encoding.....	54
7.3.2	TF-IDF .....	54
7.3.3	Word2Vec .....	56
7.3.4	FastText.....	57
7.3.5	BERT .....	58
8	Modelos .....	59
8.1	Machine learning (ML) .....	61
8.1.1	Support Vector Machine (SVM).....	63
8.1.2	Random Forest.....	65
8.1.3	XGBoost.....	68
8.1.4	Conclusiones modelos ML.....	70
8.2	Deep Learning (DL) .....	72
8.2.1	Multi-Layer Perceptron (MLP) .....	74

8.2.2	Recurrent Neural Network (RNN).....	76
8.2.3	MLP con BERT .....	77
8.2.4	MLP con BETO .....	85
8.2.5	RNN con BETO .....	87
8.2.6	RNN + MLP .....	90
8.2.7	RNN + MLP por separado .....	96
8.2.8	RNN + MLP con emojis codificados con one-hot encoding .....	99
8.2.9	Clasificación de plataforma .....	102
8.3	Replicación de estudios .....	105
8.3.1	Entorno de replicación.....	105
8.3.2	Papers replicados.....	115
9	Etiquetaje de tweets mediante LLM .....	118
9.1	Capa de Kendall .....	120
9.2	Comparativa de etiquetaje de LLM .....	121
9.3	Evaluación LLM .....	123
10	Coste temporal y económico .....	125
10.1	Coste temporal .....	125
10.2	Coste económico .....	126
11	Líneas futuras .....	127
12	Conclusiones.....	128
13	Bibliografía .....	130
14	Tablas .....	134
14.1	Tabla de ilustraciones .....	134
14.2	Tabla de tablas .....	136
14.3	Tabla de ecuaciones .....	136
15	Apéndice .....	137
15.1	Tabla de estudio del estado del arte .....	137
15.2	Criterios de clasificación de Blanquerna .....	142
15.3	Criterios de clasificación de Pere Vidal.....	144
15.4	Artículo CCIA edición 27 <sup>a</sup> .....	145

# 1 Introducción

A finales de los años 60, Estados Unidos creó un proyecto desde el departamento de defensa para crear una red de comunicación que permitiera enviar mensajes de manera rápida y eficiente a grandes distancias y así garantizar que las comunicaciones militares siguieran disponibles en el caso de que empezase una guerra nuclear. El resultado del proyecto fue una red llamada ARPANET (Advanced Research Project Agency Network), con la cual se envió un primer mensaje el 29 de octubre del año 1969.

En el año 1990 nació la conocida World Wide Web, que introdujo las páginas web y facilitó el intercambio de información entre las personas de manera mucho más fácil.

El lanzamiento del primer IPhone en 2007 dio acceso a Internet desde cualquier sitio a muchos usuarios, incrementando el uso de Internet de manera exponencial. Al cabo de unos años, hacia el 2010-2011, se empezaron a crear las primeras redes sociales, lo que permitió que muchas personas pudieran conectar de manera remota e instantánea como nunca se había visto previamente.

Internet, los móviles y las redes sociales han supuesto una revolución sin precedentes para la humanidad. Permiten que las personas se puedan comunicar instantáneamente desde cualquier parte del mundo independientemente de la distancia. Esto ha permitido que los avances sean mucho más rápidos, permitiendo compartir hallazgos e información sobre cualquier tema y ha dado lugar a una revolución tecnológica que ha cambiado el mundo para siempre.

Lógicamente, todos los avances aportan ventajas y desventajas. Con la energía nuclear se ha conseguido dar luz a muchas casas a un coste más bajo que con las energías alternativas, pero la misma tecnología crea una de las armas más destructivas jamás vistas; o por ejemplo, la invención del plástico nos ha permitido descubrir un material resistente y duradero con el cual se puede crear prácticamente cualquier tipo de objeto o forma de manera sencilla, pero con el coste de que, al no poderse reciclar fácilmente, se están contaminando muchos lugares del mundo.

Las redes sociales también aportan ventajas como por ejemplo la facilidad para compartir información con personas alrededor del mundo o la posibilidad de conocer gente que no previamente no habíamos conocido jamás. Su lado negativo es que facilitan que ciertas personas puedan difamar, insultar y acosar a los demás de manera remota y fácil. Esto ha creado la necesidad de regular ciertos comportamientos para impedir que ciertas personas puedan salir perjudicadas.

Se ha intentado moderar el contenido de las redes sociales de manera manual, pero no es posible debido a la gran cantidad de datos que se suben a diario a las redes sociales. Por esto se ha visto la necesidad de automatizar el proceso. Este proyecto pretende crear una herramienta para detectar dichos malos comportamientos y evitar que tengan impacto en “el objetivo”.

## 2 Objetivo principal

El odio o toxicidad es un comportamiento adaptativo adverso, es decir, se adapta a las regulaciones y moderaciones de cada contexto. Como el uso de modismos es muy habitual, resulta muy difícil crear un sistema que pueda detectar toxicidad en distintos contextos culturales y lingüístico de manera automática.

Este trabajo se va a realizar como parte del proyecto *Disargue* en colaboración con la Universidad de Blanquerna. El objetivo principal de este proyecto consiste en analizar qué elementos componen el odio y en crear un mecanismo de detección automático para poder clasificar los textos.

Para hacer el estudio se van a utilizar tweets y comentarios de Youtube en posts de periodistas españoles durante las elecciones europeas del 2024. Al analizar estos textos se podrá ver que cantidad de odio reciben los periodistas, si el colectivo de los periodistas recibe más odio que otros colectivos investigados en otros trabajos y qué elementos componen el odio escrito en las redes sociales.

Para realizar esta investigación se empezará estudiando qué se considera odio y qué elementos lo componen de manera teórica. A continuación, se buscarán métodos de recolección de los datos de odio para encontrar la mejor manera de hacer un estudio sistemático y justo de los textos. Después, se estudiará qué métodos se han utilizado en otros trabajos hechos previamente, como los de Luís Ángel y Paula Ximena, que investigaron el odio en las redes sociales enfocado a periodistas y a mujeres durante el 8M respectivamente. Una vez comparados los resultados obtenidos con los resultados de trabajos anteriores, se probarán tanto métodos de *Deep Learning* y de *fine tuning* propios como modelos usados en artículos del estado del arte. Finalmente, se usarán modelos de lenguaje para etiquetar los datos y así poder comparar los resultados de los modelos con los datos originales.

Los modelos propios se han creado utilizando métodos que no se han usado previamente en el estado del arte, dando un enfoque nuevo al reconocimiento de toxicidad.

Como objetivo final, el proyecto va a ser un estudio detallado de los elementos que conforman el odio, un análisis de las técnicas aplicadas y sus aplicaciones en modelos para poder mejorar las predicciones.

## 3 Marco teórico

### 3.1 Definición de odio

La definición de odio o toxicidad cambia dependiendo del ámbito en el que se trata<sup>1</sup>. Se podría decir que las dos definiciones más esenciales son la ética y la legal, ya que de ambas surgen las consecuencias del discurso de odio.

La definición de odio en cuanto a los **aspectos éticos** contempla si el sistema cumple los siguientes puntos:

- Si promueve o no valores humanos.
- Si discrimina o no.
- Si se puede explicar y si es transparente.
- Si respeta la privacidad de los usuarios.
- Si garantiza la seguridad de los usuarios.
- Si hace que los usuarios se responsabilicen de sus acciones.
- Si los humanos tienen el control o no.
- Si los operadores son profesionales o no.

La definición que propone Mudit Chaudhary en cuanto a los **aspectos legales** tiene en cuenta la alta dificultad que hay en definir el odio, ya que la definición puede variar dependiendo de los distintos contextos, por lo que hacer una sola definición que pueda tener repercusiones legales es altamente complicado. Aparte de la definición, también importa el dilema de si se tiene que penalizar únicamente a la persona que ha promovido odio o si también se tiene que penalizar a las plataformas que han permitido dicho odio.

También es relevante comentar la importancia de que dichas medidas no deben afectar a la libertad de expresión. Hay varios países en los que se han creado leyes que regulan en función de la probabilidad que tiene un mensaje de causar daño.

Según la ONU<sup>2</sup>, un discurso de odio se puede definir como cualquier forma de comunicación, escrita o de comportamiento, que pretende dañar o difamar a una persona o a un grupo de personas en función de su religión, etnia, nacionalidad, descendencia, género o cualquier otro factor identificativo.

Según Chaudhary, los estados tienen la responsabilidad de vigilar el incitamiento al odio, ya que puede escalar hasta el punto de que se realicen acciones hostiles o violentas en contra de un grupo. No otorga la responsabilidad de prohibir o controlar la libertad de expresión (aún que no condenan la abolición de este derecho de manera explícita).

---

<sup>1</sup> (Mudit Chaudhary, 2021)

<sup>2</sup> (Guterres, 2019)

En otros papers, como en el de Andrew F.Sellars, “*Defining hate speech*”<sup>3</sup>, se contemplan distintas situaciones en que se ha definido el discurso de odio sin éxito, y se observan casos como el de Justice Stewart, que definió obscenidad (un mensaje de odio y un mensaje obsceno se podrían relacionar en cierto modo) como “sé lo que es cuando lo veo”.

También se contempla el fenómeno de que algunos países regulan la libertad de expresión en función de lo que se considera “verdad” en esa situación. Es decir, la definición de odio depende de lo que en el contexto social actual se considera como tolerable o no, por ejemplo: en Suiza se considera delito de odio negar que el año 1915 hubo un genocidio contra los armenios, mientras que Turquía se considera odio decir que hubo un genocidio.

Posteriormente Sellars analiza las definiciones de Mari J. Matsuda, Calvin Massey, Mayo Moran, Kenneth Ward, Susan Benesch, Bhikhu Parekh, Alice Marwick y Ross Miller. Los trabajos propuestos por las personas mencionadas van proponiendo de manera progresiva definiciones de odio que contemplan sucesivamente los trabajos anteriores. Es importante remarcar el trabajo que se ha hecho de iteración para poder definir este concepto, ya que hay el problema fundamental de que el tema es altamente subjetivo y encontrar una definición concreta es altamente complejo.

La definición más concreta define el odio en función del contenido (es decir si el simbolismo del mensaje es denigrante para la persona), de la intención (es decir con la intencionalidad de promover odio, violencia o resentimiento en contra de un grupo marginado) y del daño (es decir, si el receptor del mensaje experimenta alguna experiencia subjetiva a causa del mensaje). Aparte de estos tres puntos se acaba añadiendo un cuarto, que consiste en evaluar si el mensaje tiene intención de transmitir una información legítima o no, es decir si aporta información útil en el debate social o público.

En el artículo de María Antonia Paz, Julio Montero-Díaz y Alicia Moreno Delgado, “*Hate Speech: A Systematized Review*”<sup>4</sup> se concluye que para identificar el odio se debería tener en cuenta si la intención del mensaje es comunicar un hecho, si se ha expresado con información que lo respalda o si se ha usado “conocimiento general”.

---

<sup>3</sup> (Sellars A. F.)

<sup>4</sup> (María Antonia Paz, 2020)

### 3.2 Implicaciones éticas y sociales de los discursos de odio

En el paper de Mudit Chaudhary, Chandni Saxena y Helen Meng, “*Countering Online Hate Speech: An NLP Perspective*”<sup>5</sup>.

Para poder reducir el odio, los autores observan que hay dos métodos, el moderado proactivo y el reactivo. El reactivo consiste en aplicar “sanciones” a los usuarios que se detecte que transmiten odio. El proactivo consiste en usar técnicas de *NLP* para poder predecir si una conversación va a contener odio y, en función de si va a tenerlo o no, aplicar métodos para prevenirlo. Independientemente de los dilemas éticos que generan tanto el uso proactivo como sobre todo el preventivo, se ha generado la necesidad de automatizar el proceso de identificación de odio debido a las consecuencias psicológicas que tienen los moderadores y a la gran velocidad a la que se genera contenido en Internet en comparación a la velocidad de moderación que tienen las personas.

Argumentan que este ámbito debe ser estudiado, ya que puede causar daños individuales (ansiedad, insomnio, etc.) y daños colectivos (creación de grupos altamente polarizados, por ejemplo).

Un debate que es necesario tener como sociedad ante la necesidad de moderar dichas plataformas es hasta qué punto se debe controlar lo que pueden decir las personas. La línea que diferencia un mensaje como libertad de expresión o como odio puede llegar a ser muy ambigua. Hay casos en que claramente el emisor de un mensaje pretende dañar al receptor, pero en muchos otros casos no está tan claro, ya que el sentido del mensaje depende de la interpretación del receptor.

Moderar con la intención de evitar comportamientos depredadores es muy distinto a no permitir que la gente se exprese libremente para que no pueda ofender a las otras personas. Esto es especialmente relevante debido a que, en la actualidad, la opinión pública está altamente polarizada en muchos aspectos. Se ha llegado hasta el punto en que la crítica de ciertos temas puede crear fricción entre las personas.

Con esta reflexión se pretende expresar la gran importancia de que se debata como sociedad qué se considera odio o no, qué implicaciones tienen dichas consideraciones y por qué. Hace falta alcanzar un consenso como país para poder moldear las políticas de moderación de las distintas plataformas de manera independiente, ya que la alternativa es que estas políticas estén dictadas por una empresa que, normalmente, no comparte el mismo contexto social e histórico del lugar donde se usa la red social y, lo que es más importante, tiene como interés principal la maximización de ganancias para la empresa, no la cohesión y el bienestar social.

Precisamente por este motivo, es especialmente relevante la creación de mecanismos de detección de odio, para lo que se requiere un sistema que pueda determinar la intención del mensaje a partir de unas guías y de la forma más imparcial posible.

---

<sup>5</sup> (Mudit Chaudhary, 2021)

## 4 Procesamiento del Lenguaje Natural (NLP)

El campo de la Inteligencia Artificial (IA) está formado por distintas ramas de especialización, como robótica, procesamiento de imagen, de sonido o predicción de tendencias, entre otras. Uno de los campos que últimamente ha estado ganando más importancia es el Procesamiento de Lenguaje Natural o en inglés **Natural Language Processing (NLP)**.

Los orígenes del NLP se remontan a los años 50, con los primeros intentos de traducción automática durante la Guerra Fría para traducir textos del ruso al inglés<sup>6</sup>. El primer *chatbot* que se creó fue ELIZA, en el año 1966<sup>7</sup>. ELIZA era un *chatbot* especializado en el campo de la psicología. Daba conversación y consejos a pacientes, y su funcionamiento consistía en un conjunto de reglas que se usaban para elegir qué frase predeterminada es la más adecuada. No había comprensión del texto por parte del modelo.

La tecnología progresó con los años, hasta que se consideró que el campo del NLP se había quedado estancado. Los métodos que había en los años 90 no representaban correctamente las palabras, los modelos no entendían los textos y necesitaban órdenes prestablecidas para funcionar.

A continuación se puede ver una imagen que muestra algunos de los puntos más importantes en el campo del NLP a partir de los años 2000.

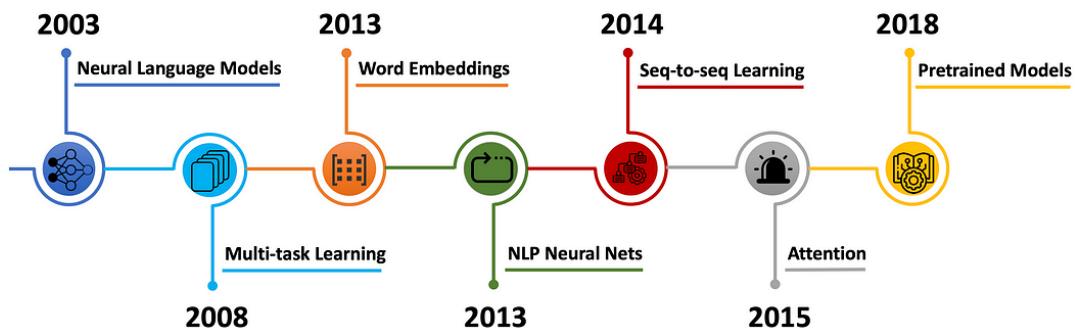


Ilustración 1: línea temporal del progreso del NLP<sup>8</sup>

En el año 2013 se inventó el sistema de codificación *Word2Vec*, que permitió representar las palabras teniendo en cuenta su significado (explicado con detalle en el capítulo “Word2Vec”). Esto hizo que la comprensión de los modelos aumentase drásticamente. En el año 2017 se creó el Transformer. Los Transformers permitieron aumentar aún más la capacidad de codificación y comprensión de las palabras. Gracias a los Transformers se consiguió crear modelos de lenguaje (LLM) que generan y entienden el texto.

<sup>6</sup> (malik, 2024)

<sup>7</sup> (Weizenbaum, 1966)

<sup>8</sup> (Louis, 2020)

## 4.1 Conceptos NLP y Ciencia de Datos

A continuación, se van a explicar conceptos del NLP y al campo de la ciencia de datos que se utilizarán durante este documento:

### **Conceptos Ciencia de Datos:**

- Modelo: procedimiento estadístico o algoritmo que extrae patrones de los datos (este concepto se explica en más detalle en el apartado “Modelos”).
  - Ejemplo: regresión lineal, SVM, red neuronal o transformer.
- Entrenamiento de modelo: proceso que hace un modelo para extraer alguna conclusión de los datos, de este modo “aprende” a realizar el objetivo.
- Fine-tuning: un modelo entrenado ha encontrado unos patrones de unos datos, se puede continuar el entrenamiento de un modelo para especializarlo o mejorarlo en un campo.
- Evaluación de modelo: proceso que consiste a poner a prueba el modelo con datos que no ha visto durante el entrenamiento para evaluar si ha aprendido correctamente.
- Partición de datos: para poder hacer el entrenamiento y evaluación del modelo se dividen los datos para así utilizar unos para entrenar y los otros para evaluar (este concepto se explica en más detalle en el apartado “Modelos”).
- Overfitting: si un modelo aprende “de memoria” los datos, no sabe hacer predicciones correctamente ya que no ha generalizado su aprendizaje, a este fenómeno se le llama overfitting o sobreentrenamiento.
- Underfitting: fenómeno contrario al overfitting. Consiste en que el modelo no ha aprendido bien el patrón y no ha aprendido suficiente.
- Métricas: al evaluar un modelo se puntúa su rendimiento mediante el cálculo de una métrica o puntuación.
  - Ejemplo: accuracy, f1 score o recall.
- Varianza: medida que indica la dispersión o variabilidad de un conjunto de valores respecto la media. En este trabajo, se han ejecutado múltiples veces los modelos para poder obtener estimaciones robustas de las métricas. Para cada métrica se ha calculado la media con 10 valores y su varianza. En los resultados de los modelos se puede ver que se indica primero el valor de la métrica, luego se usa el símbolo “±” y finalmente se indica la varianza de la métrica.
- Puntuación ROC-AUC: puntuación que mide como de bien clasifica un modelo en un problema binario. Su valor va del 1 al 0, 1 siendo puntuación perfecta, acierta todo y 0.5 hace predicciones aleatoriamente. También se puede representar con una curva, facilitando su interpretación.
- Predicción/inferencia: conclusión a la que llega el modelo con unos datos nuevos, en este caso se predice si un tweet o un comentario contiene toxicidad o no.

### Conceptos NLP:

- Token: es una unidad textual que puede representar una palabra, una subpalabra o una letra. Son los elementos que se utilizan para poder tratar las palabras.
  - Ejemplo: “Café”, “a”, “veinte” o “tiesto”.
- Tokenizar: proceso que convierte un texto en una lista de tokens.
- Documento: conjunto de tokens que forman una pieza coherente de texto.
  - Ejemplo: un texto, un tweet o un comentario.
- Vocabulario: lista de tokens únicos que entiende el modelo.
- Corpus: colección de documentos que se utiliza para entrenar el modelo.
- N-grama: secuencia de n tokens.
  - Ejemplo: “buenos días” (bigrama) o “Trabajo final de máster” (quagrama)
- Stemming: reducción de una palabra a su raíz (este concepto se explica en más detalle en el apartado “One-hot encoding”)
  - Ejemplo: “cantarás” → “cant”
- Lemmatization: reducción de una palabra a su lemma (este concepto se explica en más detalle en el apartado “One-hot encoding”)
  - Ejemplo: “cantarás” → “cantar”
- Stop-words: palabras funcionales muy frecuentes en el texto que normalmente añaden ruido (este concepto se explica en más detalle en el apartado “One-hot encoding”)
  - Ejemplo: “a”, “el” o “de”.
- Modelo de lenguaje: modelo especializado en la predicción en el campo del NLP, se utilizan para clasificar, generar textos, generar código y sintetizar texto entre otras funcionalidades (este concepto se explica en más detalle en el apartado “Etiquetaje de tweets mediante LLM”).
  - Ejemplo: GPT-3, Gemini o Meta-Llama-3.
- Embedding: representación de una palabra en formato numérico o vectorial. Se necesita codificar las palabras para que los modelos puedan utilizarlas para hacer predicciones (este concepto se explica en más detalle en el apartado “Encodings y embeddings”).

Los conceptos comentados previamente se utilizan durante el documento de forma regular, no obstante, se explican otra vez y en más detalle en sus apartados correspondientes.

## 4.2 Problemas con NLP para la detección de odio

Según un estudio realizado por la Universidad Pompeu Fabra (Barcelona) y la Universidad Simon Fraser (Vancouver)<sup>9</sup>, los retos encontrados hasta ahora en el área de la detección de odio mediante técnicas de NLP muestran que es necesario cambiar ciertas convenciones y prácticas que se han estado usando para la clasificación de textos como odio. En las conclusiones de su trabajo indican que los métodos actuales no son suficientes para detectar el odio sin afectar a minorías marginales. A continuación, se redactarán la metodología actual y las mejoras que proponen:

- Definición de discurso de odio: la definición depende de la cultura y de la política de la zona donde se está investigado el odio. Esto añade una capa extra de interpretabilidad y de contexto en los textos.
- Tipos de odio: las técnicas actuales son buenas detectando las categorías más comunes (por ejemplo, el insulto) lo cual hace que no se detecten bien los mensajes. El odio puede venir inherente en el contexto, por lo cual las técnicas NLP, que solo se centran en el texto, no lo detectan.
- Intencionalidad en el etiquetaje: al no haber comunicación entre el analista y el emisor del mensaje, no se puede saber con certeza la intencionalidad por lo cual se tiene que intuir por el analista (por lo cual se añade también sesgo por parte del anotador).
- Sistema de anotación: normalmente se etiqueta el texto asumiendo que solo hay una respuesta válida. Para poder etiquetar los mensajes de odio entre distintas personas es necesario crear un documento *Inter-Annotator Agreement (IAA)* para así tener una pauta con lo que se considera una categoría u otra.
- Las técnicas actuales de NLP dependen de correlaciones falsas<sup>10y11</sup>, esto hace que, al eliminarse estas relaciones, se baje la precisión del modelo.
- Generalización de los modelos: los modelos aprenden las relaciones entre tokens, pero no aprenden a generalizar el concepto de odio.
- Dado que el odio solo aparece de manera irregular en ejemplos reales, los datasets que se crean para los estudios son generados de manera artificial, esto hace que no puedan aparecer conclusiones parecidas en estudios separados, por lo que no se crean dataset complementarios.

---

<sup>9</sup> (Paula Fortuna)

<sup>10</sup> (Aymé Arango, 2019)

<sup>11</sup> (Md Mustafizur Rahman, 2021)

### 4.3 Creación de un dataset para el estudio del odio

Un grupo de investigación en Texas hizo un estudio llamado “*An Information Retrieval Approach to Building Datasets for Hate Speech Detection*”<sup>12</sup> en el cual remarcan aspectos importantes en el momento de crear un *dataset* de datos para hacer estudios de odio con métodos de NLP.

Es de esencial importancia tener en cuenta ciertos aspectos en el momento de diseñar un *dataset* ya que el odio no es un tema fácilmente representable, es decir, hay que tener en cuenta que los mensajes de odio representan un porcentaje muy pequeño del total de los datos (si se hace *subsampling* es posible que solamente se entrene con datos sin odio); también se tiene que tener en cuenta que el odio no es siempre del mismo tipo, los insultos son fáciles de detectar ya que solo se tiene que identificar un *corpus* de palabras, las críticas y mensajes irónicos son mucho más difíciles de detectar ya que necesitan contexto e interpretación; etc.

En su estudio recolectan tweets (sin aplicar ningún filtro) para formar un *corpus* de documentos (tweets), luego, los etiquetan mediante dos métodos *pooling* y *Active Learning* (AL). En los métodos dos métodos se empiezan por elegir aleatoriamente partes de los datos. El siguiente paso depende del método:

- *Pooling*: se utilizan métodos *ensamble* para poder determinar que tweets tienen más posibilidades de ser discurso de odio para posteriormente agruparlos. Una vez se han creado las agrupaciones se etiquetan los twits manualmente por personas, haciendo que las personas solo tengan que revisar los mensajes más “sospechosos”.
- *Active Learning (AL)*: consiste en usar la técnica de boosting. Se entrena el modelo con un set de datos, las personas corrigen manualmente el resultado y se vuelve a entrenar el modelo con los datos correctos, se repite el proceso hasta que el modelo haya aprendido a clasificar correctamente.

Al comparar los dos métodos se llega a la conclusión que, para este caso, funciona mejor *active learning* ya que consigue clasificar correctamente el 80% de los tweets por la mitad de coste que con la técnica de *pooling*.

Una vez entrenados los modelos evalúan si se ha conseguido entrenar a modelos que detecten el odio. Para hacer eso realizan dos experimentos, el primero con mensajes sin palabras de odio y otro experimento con mensajes con palabras que demuestran odio. La conclusión es que los modelos actuales dependen mucho de las palabras para poder detectar el odio, la precisión de los modelos baja drásticamente.

---

<sup>12</sup> (Md Mustafizur Rahman, 2021)

Method	Class	Without Hate Words			With Hate Words		
		P	R	F1	P	R	F1
BiLSTM [1]	Non-Hate	95.34	97.61	96.47	84.98	86.29	85.63
LSTM [10]	Non-Hate	95.60	96.25	95.93	84.95	89.38	87.11
BERT [27]	Non-Hate	96.25	96.25	96.25	88.84	83.01	85.82
BiLSTM [1]	Hate	12.50	6.666	8.695	36.03	33.61	34.78
LSTM [10]	Hate	15.38	13.33	14.28	40.21	31.09	35.07
BERT [27]	Hate	26.66	26.66	26.66	42.48	54.62	47.79

Ilustración 2: métricas modelos entrenados con y sin palabras de odio

Para mejorar los modelos sugieren que se entrene un conjunto de modelos de clasificación, luego combinar los resultados con *staking* y finalmente aplicar *random sampling* para distribuir los datos y que así se distribuían los datos de manera equitativa y aleatoriamente en todos los modelos.

Uno de los puntos importantes que mencionan como posibles líneas de estudio futuras es que un insulto puede variar dependiendo del receptor del mensaje.

#### 4.4 Métodos para la detección del odio con NLP

Las técnicas de *Data Augmentation (DA)* son métodos para generar datos sintéticos mediante datos reales. Se pueden aplicar a muchos campos, como NLP, imágenes, forecasting, etc. Modifican los datos para poder crear nuevos y conseguir que el modelo aprenda a generalizar mejor y que sea más robusto. Este método puede ser especialmente eficaz para detectar patrones de odio con palabras distintas y variaciones de texto.

En el estudio de Md Saroor Jahana, Mourad Oussalaha, Djamila Romaissa Beddiaa, Jhuma kabir Mim y Nabil Arhaba, “A Comprehensive Study on NLP Data Augmentation for Hate Speech Detection: Legacy Methods, BERT, and LLMs”<sup>13</sup> se investigan diversos métodos de aumentación de textos en el contexto del odio. La intención del estudio es encontrar un método fiable de aumentar los pocos datos de odio que hay y así poder entrenar modelos con más datos.

Los métodos que investigan son los siguientes: *Thesaurus-guided Synonym Substitution*, *Word-Embedding Synonym Substitution*, uso de modelos *transformer*, *back-translation (BT)*, *paraphrasing*, *MixUp for Text*, *Instance Crossover Augmentation*, *Large Language Models (LLM)*, *blank noising*, *QWERTY Keyboard Error Injection*, *unigram noising*, *sentence shuffling*, *random insertion*, *random swap*, *random deletion*, *WordNet*, *semantic embeddings*, *BERT Mask*, *random spelling error injection* y *back noising*.

En este estudio se concluye que los diversos métodos probados de *Data Augmentation* consiguen incrementar la cantidad de datos de manera exitosa. El mejor método que se consigue es la combinación de *Back-Translation* junto a BERT y Cosine. También se remarca que los *large language models (LLM)* consiguen unos resultados muy buenos, pero tienen como punto débil el alto coste computacional que se necesita para que puedan realizar su tarea.

<sup>13</sup> (Md Saroor Jahana, 2017)

## 5 Estado del arte

En la Universidad Autónoma de Madrid, en colaboración con el gobierno de España, se ha hecho un estudio llamado “Detecting and Monitoring Hate Speech in Twitter”<sup>14</sup> en el cual se consigue crear un método de detección del odio mediante el uso de redes neuronales LTSM + MLP con una puntuación de 82.8% en la curva AUC.

Para poder obtener los datos se hace un proceso en el cual se leen los tweets y se busca si contienen palabras de odio absoluto o relativo (es decir, palabras que dependiendo del contexto pueden ser consideradas odio o no, como por ejemplo “negro”, que puede ser utilizada como color o para referirse a una etnia). Si contienen odio absoluto se guarda directamente, si se trata de odio relativo se busca si tienen insultos genéricos o no.

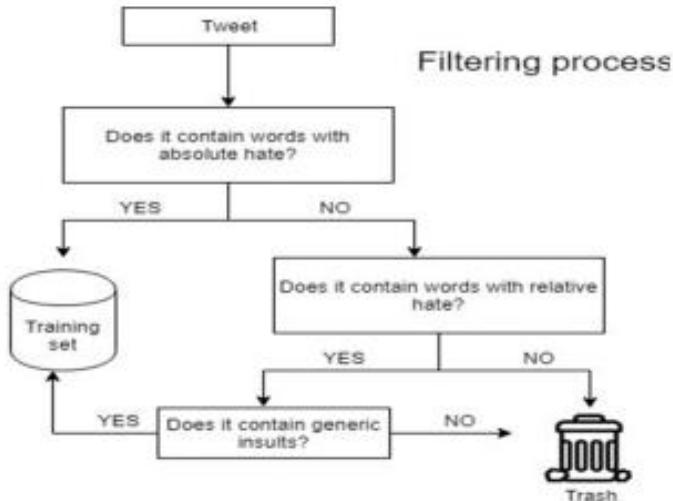


Ilustración 3: Proceso de filtraje de mensajes de odio

Una vez filtrados los tweets, se hace un proceso de etiquetación manual para poder clasificar si realmente se trata de un mensaje de odio o no.

Una vez obtenidos los tweets, se hace un proceso de tokenización para poder representarlos correctamente. Se contempla que la lengua española tiene el problema de que se puede utilizar composición de palabras, por lo que una palabra que no transmite odio se convierte en una que sí que lo hace. Por ejemplo: “hotel” es una palabra normal, pero al añadir “-UCHO” se convierte en una palabra despectiva.

---

<sup>14</sup> (Juan Carlos Pereira-Kohatsu, 2019)

En el estudio también se contempla la gestión de los emojis, para poder extraer contexto de ellos, y se unifican expresiones como “jaja”, “jaj”, “jajaj”, etc. como un mismo token para dar a la expresión un único valor.

Para poder representarlos en el estudio se utiliza TF-IDF de modo que los tweets aparezcan en forma de vector y se mantenga el orden de las palabras, que es fundamental para extraer el significado del mensaje; no es lo mismo “pobre hombre desafortunado” que “hombre pobre desafortunado”.

Finalmente, se ha planteado que los datos de los tweets de odio se representen en un grafo para posteriormente poder aplicar PageRank y así localizar los nodos que más odio representan y los que más reciben.

El flujo de procesamiento final que se plantea es el siguiente:

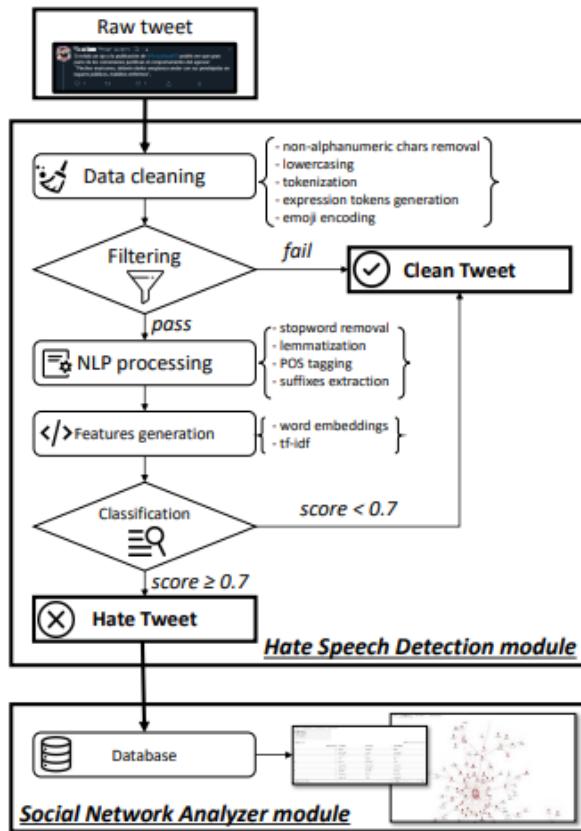


Ilustración 4: Arquitectura implementada en estudio UAM

Parte de la investigación hecha para poder enfocar el proyecto ha consistido en investigar distintos estudios que se han hecho previamente, tanto en LaSalle como en otros centros de investigación, para determinar cuál es el mejor enfoque para el problema que se pretende solucionar y continuar a partir de sus conclusiones.

A continuación se muestran los resultados de dicho estudio:

## 5.1 Resumen estudios previos

Este apartado es el resultado del análisis de distintos trabajos previos. La investigación realizada es mucho más amplia y para simplificar el texto se ha escrito el resumen que se puede ver a continuación. La investigación original se encuentra en el apartado “Tabla de estudio del estado del arte”.

A aspectos generales, las observaciones más relevantes son:

- El léxico y leyes de cada región son importantes.
- Tener en cuenta los emojis para ganar contexto.
- Usar técnica se undersampling o oversampling para evitar el desbalanceo.
- El paper “A systematic review of hate speech automatic detection using natural language processing” tiene más datasets de HS etiquetados.

En cuanto a los modelos, los mejores de cada estudio son:

- TFG Luis Ángel: Logistic Regression (aprox. 92%)
- TFG Paula Ximena: LSTM (aprox. 90%)
- TFM Oriol Ramis: Random Forest (aprox. 99%)
- Artículo 1: LSTM + MLP (threshold 0.7) (AUC 0.828)
- Artículo 2: RNN
- Artículo 3: CNN (aprox. 91%)
- A systematic review of hate speech automatic detection using natural language processing:
  - "A Survey on Hate Speech Detection using Natural Language Processing" by Schmidt and Wiegand (2017)
    - Modelos: SVM (90%), Logistic Regression (92%), CNN (88%) y RNN (87%)
  - "A Survey on Automatic Detection of Hate Speech in Text" by Fortuna and Nunes (2018)
    - Modelos: Logistic Regression (91%), SVM (89%), Naïve Bayes (86%) y Ensemble methods (93%)
  - "Cyberbullies in Twitter: A Focused Review" by Tsapatsoulis and Anastasopoulou (2019)
    - Modelos: SVM (88%)

- "Detection of Hate Speech in Social Networks: A Multilingual Perspective" by Al-Hassan and Al-Dossari (2019)
  - Modelos: SVM for multilingual hate speech detection (90%)
- "Detection of Hate Speech: A Comprehensive Review" by Mishra et al. (2019)
  - Modelos: CNN (90%) y LSTM (90%)
- "Resources and Benchmark Corpora for Hate Speech Detection: A Systematic Review" by Poletto et al. (2020)
  - Modelos: SVM (88%), Logistic Regression (87%), CNN (89%) y RNN (90%)
- "Tackling Online Abuse: A Survey of Automated Abuse Detection Methods" by Pradhan et al. (2020)
  - Modelos: Logistic Regression (90%), SVM (89%) y métodos deep learning (92%)
- "Towards Generalisable Hate Speech Detection: Obstacles and Solutions" by Yin and Zubiaga (2021)
  - Modelos: BERT (91%), CNN (88%) y RNN (90%)
- "A Review on Offensive Language Detection and Hate Speech Datasets" by Alkomah, Fatimah, and Ma (2022)
  - Modelos: CNN (90%) y RNN (91%)
- "A Review on Pre-processing Techniques for Short-text Quality Improvement in Hate Speech Detection" by Naseem et al. (2021)
  - Modelos: SVM (88%) y modelos de deep learning models (90%)
- "A Survey of Recent Neural Network Models on Code-Mixed Indian Hate Speech Data" by Dowlagar, Suman, and Mamidi (2021)
  - Modelos: CNN (89%), RNN (91%) y BERT (92%)

## 6 Datos

### 6.1 Datasets

Los datasets que se han utilizado para este trabajo vienen de dos fuentes principales, de Twitter y de Youtube.

Los comentarios de la plataforma de Twitter provienen del Trabajo de Final de Grado de Luís Ángel<sup>15</sup>. En dicho trabajo utilizaron la API de Twitter para poder recolectar un total de 198.240 Tweets relacionados con los periodistas que la Universidad de Blanquerna indicó. Una vez recolectados, Blanquerna, hizo un etiquetaje manual.

El resultado fue un dataset con 41.780 tweets, de los cuales 39.833 fueron clasificados como “sin odio” y 1.947 como “con odio”. Esto significa que el 4.66% de los datos encontrados se contienen odio, haciendo que el total de tweets que se pueden utilizar para la clasificación sea de 3.894 (teniendo un 50% de cada categoría).

Los comentarios de la plataforma de Youtube se han recolectado mediante la API de Youtube (la explicación del proceso de recolección se puede ver en el capítulo “Script Youtube”). Se recolectaron un total de 54.146 comentarios y la Universidad de Blanquerna hizo un etiquetaje manual.

El resultado del etiquetaje fue un dataset con 22.501 comentarios, teniendo 20.697 clasificados como que no contienen odio y 1.804 comentarios clasificados como que sí que contienen odio. Esto significa que el 8.72% de los comentarios contienen odio. Al hacer un dataset con un balance del 50% con los comentarios se consiguen un total de 3.608 datos.

Esto hace que el total de los datos disponibles (si se integran los datos de los dos datasets en un mismo corpus de comentarios) sea de 7.502 comentarios.

---

<sup>15</sup> (Servera, 2024)

## 6.2 EDA

En este capítulo se verá el análisis de los comentarios que se ha hecho para poder ver la naturaleza de los datos.

Los datos analizados están compuestos de comentarios de YouTube y de tweets, se analizarán tanto por junto como por separado.

### 6.2.1 Estudio de los textos

Los comentarios están compuestos tanto de texto como se emojis, en este apartado se analizará la parte de los textos.

Previamente al análisis tradicional del área del NLP como sería el conteo de palabras más usadas o representaciones de wordclouds, se hizo un análisis de las categorías que aparecían más en los datos, es decir, buscar temáticas en función de las palabras.

En la parte derecha de las gráficas se pueden ver las categorías, se puede ver que las palabras no están completas, esto es debido a que se han buscado las raíces que representan la categoría, por ejemplo, Europa se ha buscado como euro o palestina como pales.

Como el estudio se está haciendo en el contexto de las elecciones europeas del año 2024, los temas elegidos son de tipo político i regional.

El primer tema ha sido por países, los elegidos son, Europa, USA, China, Israel y Palestina. La elección de estas regiones ha sido para investigar si el contexto geopolítico era un tema de conversación en el período de las elecciones europeas.

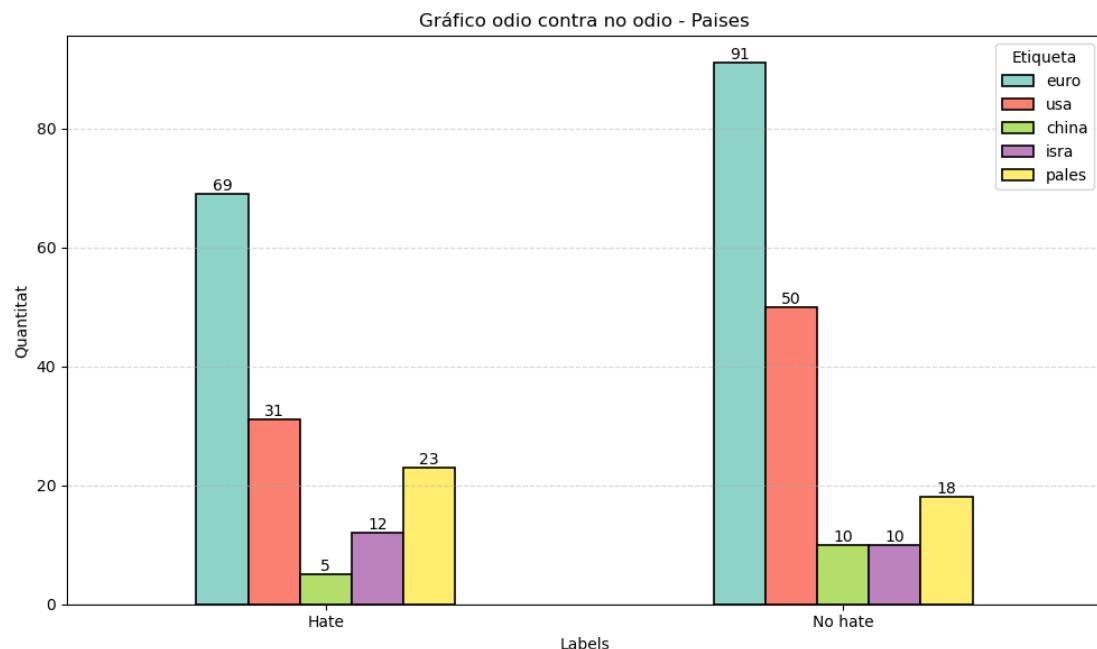


Ilustración 5: gráfica de odio separado por zona geográfica

Como se puede ver en el gráfico, las categorías de Europa, USA y China no reciben un incremento de odio entre las fases de odio y no odio, mientras que Israel tiene un incremento del 27% de odio y palestina del 20%. Esto indica que en estos períodos de tiempo hay un incremento de odio dirigido hacia el conflicto más reciente, mientras que los temas más centrales al contexto geopolítico en general son menos relevantes.

El sistema de voto para el parlamento europeo funciona por grupos de partidos regionales organizados en grupos europeos. Como el contexto es España se decidió analizar los distintos partidos políticos para ver si hay un incremento de odio hacia ellos o no y para ver cual tiene más presencia. El resultado es el siguiente:

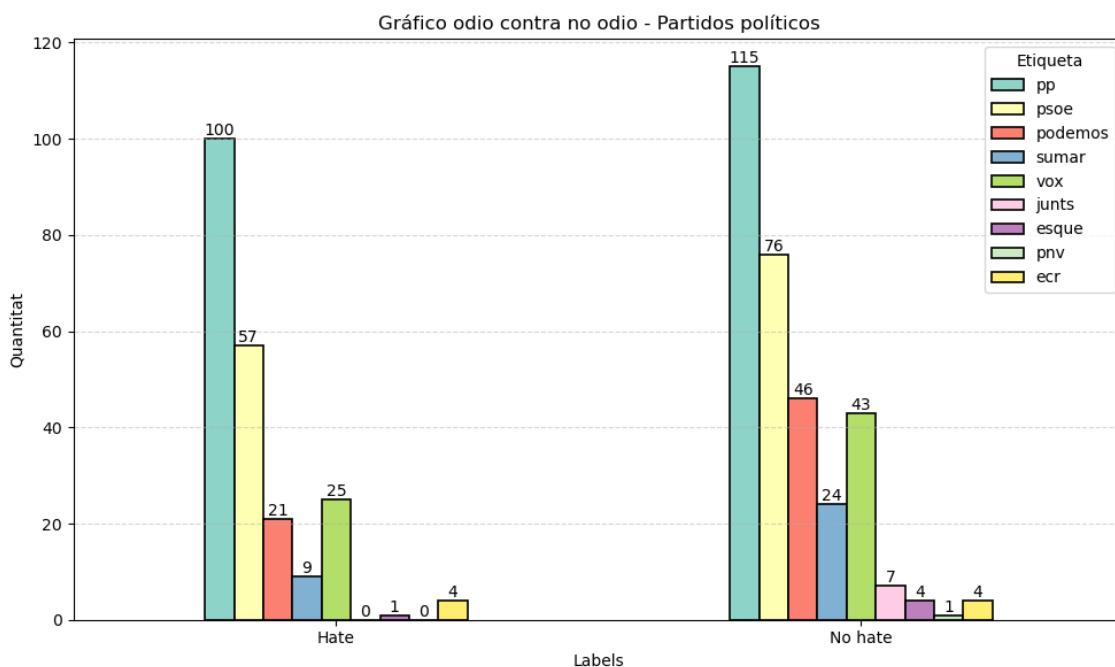


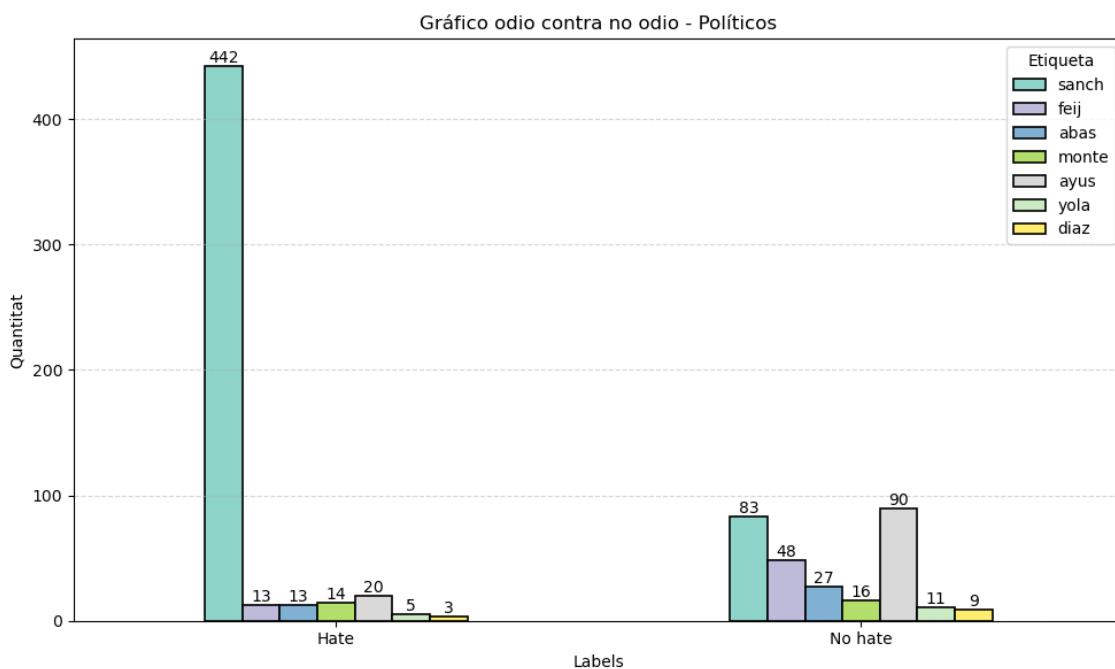
Ilustración 6: gráfica de odio por partido político

Mirando los resultados del estudio del odio se puede ver como la presencia de las letras PP muy alta cuando, esto se puede deber a dos factores, primero la mayor presencia que tiene la derecha en las redes sociales o a que el partido popular europeo tiene casi las mismas siglas que el partido popular (PPE y PP) mientras que los otros partidos no tienen nombres parecidos, por ejemplo, el PSOE se presenta en el grupo europeo S&E o Sumar i ERC que se presentan em el grupo de Los Verdes.

Para comprobar esta teoría se hizo una clasificación en función de los nombres de los partidos europeos. Al clasificar el odio en función del grupo europeo no se encontró ninguna mención en específico a ninguno de los grupos: S&D, EPP, ECR, Verdes, Renew Europe, The Left, ID y ALE.

Esto indica dos cosas, primero que la hipótesis de que el PP tiene más relevancia debido a que la gente menciona al PPE es falsa y segundo que en las elecciones europeas, los partidos nacionales tienen mucha más relevancia que los europeos.

Viendo la alta relevancia de la política española en el contexto de los comentarios, se decidió volver a la política nacional, si se investiga que políticos reciben más odio se puede ver lo siguiente:



*Ilustración 7: gráfica de odio por políticos españoles*

Se puede ver que el político que más presencia tiene es Pedro Sánchez, esto sorprende porque, aun teniendo en cuenta que es el presidente del gobierno (al ser el más influyente de todos, puede ser que reciba más odio), el incremento de odio recibido tiene un incremento del 436% entre no odio y odio, esto es sorprendentemente alto si se compara con el odio que recibe el partido del PSOE, siendo de 57 comentarios. Esto indica que en este caso el odio es personal en vez de colectivo, indicando un cambio de tipología de odio.

Otro punto que llama la atención es que la diferencia de odio de Alberto Núñez Feijóo es negativa, pasando de 48 comentarios a 13, si se mira Ayuso se puede ver qué pasa lo mismo que con Feijóo, el odio se reduce de 90 comentarios a 20. Esto indica que en este caso el odio es personal en vez de colectivo, indicando un cambio de tipología de odio.

Esta gran diferencia es indicativa de que claramente los comentarios tienen intencionalidad política ya que, teniendo en cuenta que los dos son los líderes de los dos partidos principales de España, tendrían que recibir cantidades de odio parecidas ya que son muy conocidas e influyentes. Aparte de la influencia que tienen los dos, es importante destacar que se ha encontrado que el PP tiene más presencia que el PSOE en las redes

sociales, indicando que parte de los comentarios son seguramente de gente no votante del PSOE.

Siguiendo por la misma rama de estudio de los políticos españoles, se decidió investigar el odio recibido por los distintos políticos españoles, pero esta vez a los que se presentan a la eurocamara. Los resultados se pueden ver en la siguiente gráfica:

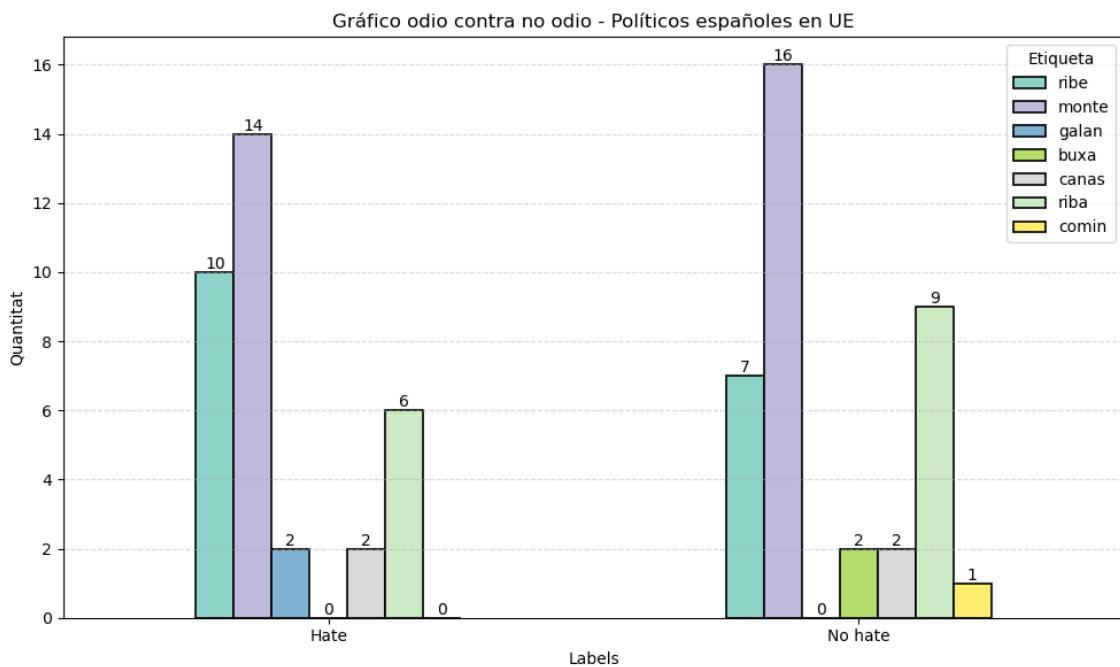


Ilustración 8: gráfica de odio por políticos europeos

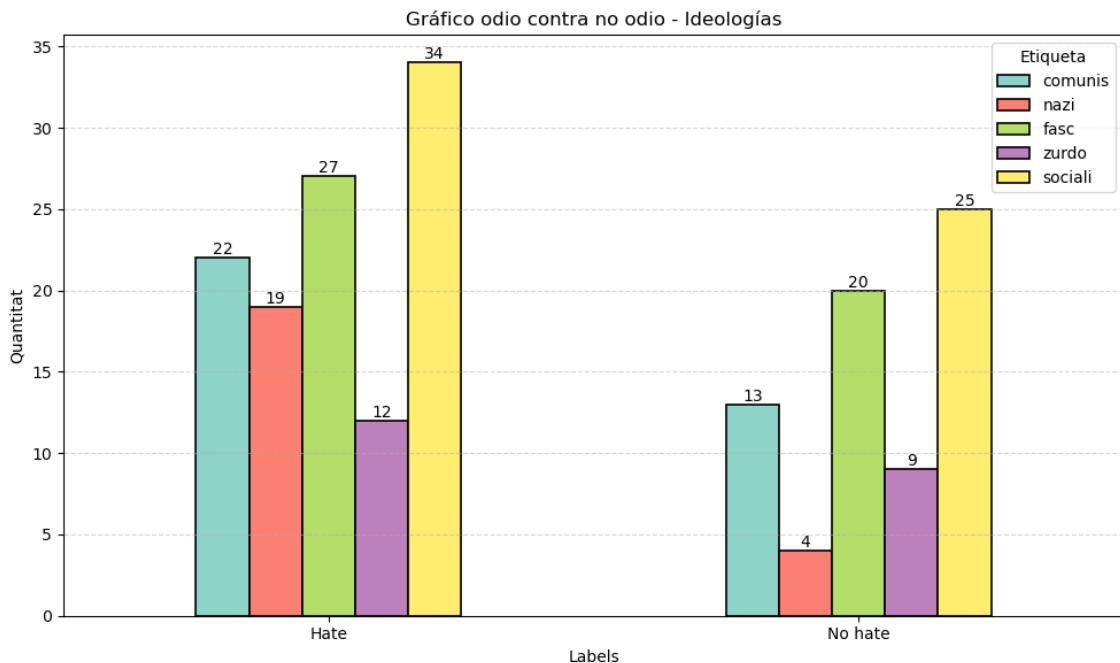
Los políticos que se han buscado son los siguientes:

- Ribe: Teresa Ribera (PSOE)
- Mont: Dolors Montserrat (PP)
- Galan: Estrella Galán (Sumar)
- Buxa: Jorge Buxadé (Vox)
- Canas: Jordi Canas (Ciudadanos)
- Riba: Diana Riba (Ara Repùblicas)
- Comin: Toni Comín (Junts)

Se puede ver que Teresa Ribera recibe un incremento de odio del 42%, Dolors Montserrat pasa de tener 14 comentarios de odio a 16 de no odio, Estrella Galán pasa de no recibir odio a recibir 2 comentarios clasificados como odio, Jorge Buxadé pasa de tener 2 comentarios clasificados como odio a ninguno si odio, Jordi Canas no incrementa sus comentarios de odio, Diana Riba tiene 6 comentarios de odio y 9 de no odio y Toni Comín solo tiene un comentario que no contiene odio.

Después de investigar a los políticos europeos que se presentan a la eurocamara se decidió investigar la presencia de las distintas ideologías políticas en los comentarios.

Si se buscan las distintas ideologías se puede ver lo siguiente:



*Ilustración 9: gráfica de odio por ideología*

Se puede ver que la única ideología que no recibe un incremento entre odio y no odio es “fascista” mientras que todas las otras aumentan.

La ideología que tiene un incremento más alto es “Nazi” con un incremento del 375%, luego “Comunista” con un incremento de 69%, “Socialista” con un 36% y finalmente “Zurdo” con un incremento del 33%.

Visto que las palabras que identifican una orientación política se usan claramente para transmitir odio, se decidió investigar si hay diferencia entre las regiones de España. Como cada comunidad autonómica tiene un gobierno de una ideología distinta, se esperaría encontrar un incremento sustancial entre las comunidades regidas por los distintos partidos políticos.

Para hacer esto se decidió investigar cuánto odio recibe cada comunidad autónoma e investigar qué partidos gobiernan.

Los resultados se pueden ver en la siguiente gráfica:

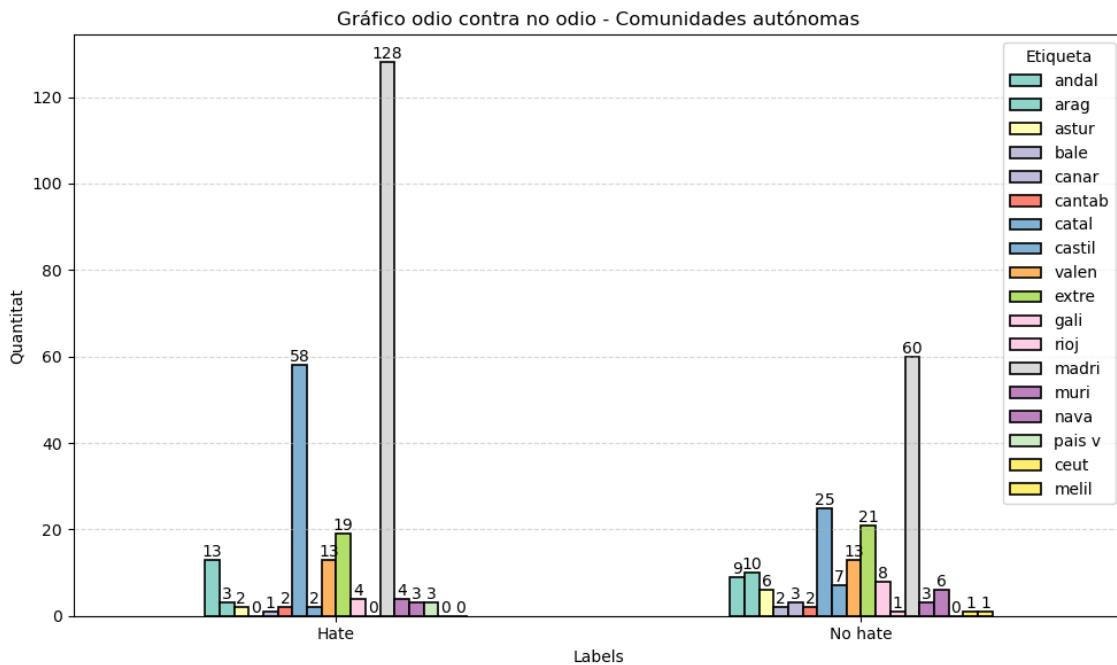


Ilustración 10: gráfica de odio por comunidad autónoma

Si observamos la gráfica anterior podemos ver que las comunidades que reciben un incremento más sustancial entre no odio y odio son:

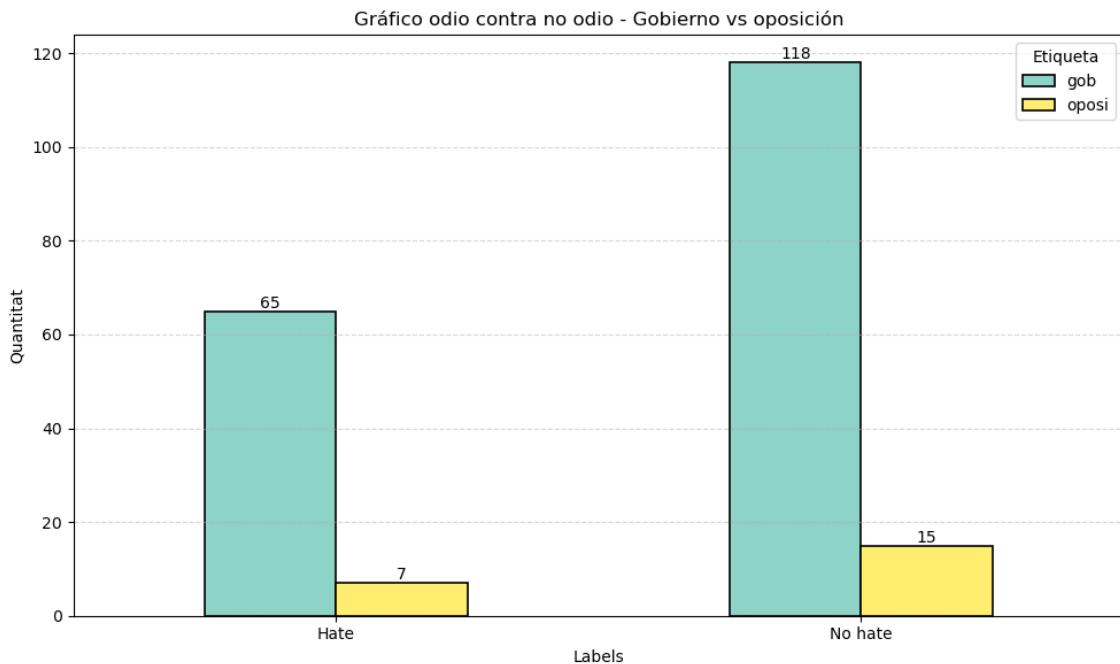
- Andalucía: incremento de 9 a 13 (44%)
- Catalunya: incremento de 25 a 58 (132%)
- Madrid: incremento de 60 a 128 (113%)

Viendo estos resultados puede parecer que Catalunya aparece muchas más veces, esto podría ser por dos motivos, el primero siendo que en el contexto de la política española, Catalunya es un tema que tiene controversia (de allí teniendo mucho más odio que las otras comunidades) o que las elecciones europeas y las del parlamento catalán se encontraban cerca, las europeas siendo entre el 6 y el 9 de junio y las catalanas siendo el 12 de mayo (al estar siendo el mismo año puede ser que siga habiendo debate).

Al igual que con Catalunya, Madrid también tiene un incremento muy alto de odio, esto puede ser porque, al igual que en Catalunya, el entorno político es muy activo, haciendo que se produzca mucho debate alrededor de ambas comunidades.

Si se investigan cuando fueron las últimas elecciones al parlamento de Andalucía y al parlamento de Madrid se puede ver que fueron en el año 2023, descartando que el debate venga de las últimas elecciones.

Finalmente, se decidió investigar qué diferencia hay entre comentarios de odio y no odio que mencionan al gobierno y que mencionen a la oposición.



Se puede ver cómo ni gobierno ni oposición reciben más odio que comentarios de no odio, lo que sí que se puede apreciar es que proporcionalmente el gobierno tiene mucha más presencia en las redes que la oposición.

Si se calcula la diferencia de comentarios de manera porcentual se puede ver que son relativamente parecidas, la categoría “Gobierno” cae un 55% mientras que la categoría “Oposición” cae un 53%.

Vistos los temas que más aparecen en las plataformas analizadas se investigaron qué palabras son las más frecuentes. Para poder representar los resultados de las palabras más frecuentes se han utilizado dos tipos de gráficas, wordclouds y gráfica de barras.

Las gráficas de wordcloud son gráficas que muestran un conjunto de palabras dispersadas por la gráfica. Cada palabra tiene un color (para diferenciarlas entre sí) y un tamaño distinto. El tamaño indica la cantidad de veces que aparece una palabra en todo el vocabulario (ignorando las stopwords, explicación del concepto en el capítulo “One-hot encoding”) de palabras, cuanto más grande la palabra aparece más frecuencia es.

Para empezar el análisis se decidió mostrar las palabras más frecuentes de todo el vocabulario de los datos.

La siguiente gráfica wordcloud muestra las palabras más usadas en todos los datos, independientemente de la plataforma y de la etiqueta asociada.



Ilustración 11: wordcloud de todas las palabras

Si se miran las palabras más relevantes en el total de datos se puede ver que son “gente”, “va”, “solo”, “España”, “puede”, “ver” y “bien”.

A simple vista no se puede llegar a ninguna conclusión de que importancia tienen las palabras más importantes ya que en el wordcloud anterior se están mostrando las palabras de todas las plataformas y sin separar por odio o no odio.

Las únicas palabras que destacan y que se podría intuir algún significado son “España”, “periodista”, “mujer” y “hombre”.

Para poder extraer conclusiones de las posibles razones por las que estas palabras son tan frecuentes se va a hacer un estudio separando las palabras por plataforma y analizándolas con wordclouds y con gráficos de barras (para poder ver su frecuencia de manera numérica). Los elementos de color rojo representan la plataforma Youtube mientras que los elementos azules representan a Twitter.

Las siguientes gráficas estudian las plataformas sin tener en cuenta el odio, se muestran todas las palabras del vocabulario.



Ilustración 12: wordcloud de todas las palabras separadas por plataforma

Se puede ver que las palabras que predominan en el vocabulario dependen de la plataforma, en Twitter son: "ahora", "España", "Madrid" y "periodista" mientras que en Youtube son: "gente", "España", "gracias" y "solo".

Se puede ver que las palabras más frecuentes en Twitter parecen ser más orientadas a un entorno más político, como es normal en la plataforma, mientras que Youtube tiene palabras más genéricas.

Para poder visualizar fácilmente la cantidad de palabras que hay en cada plataforma se han representado los datos en una gráfica de barras donde se muestra en gris el total de veces que aparece una palabra y dentro de dicha barra se encuentran dos barras más, una azul que indica la cantidad de veces que aparece esa palabra en Twitter y una línea roja que muestra la cantidad de veces que aparece la palabra en Youtube.

Para analizar bien la cantidad de palabras se ha decidido que se visualizaran las 30 palabras más frecuentes del vocabulario.

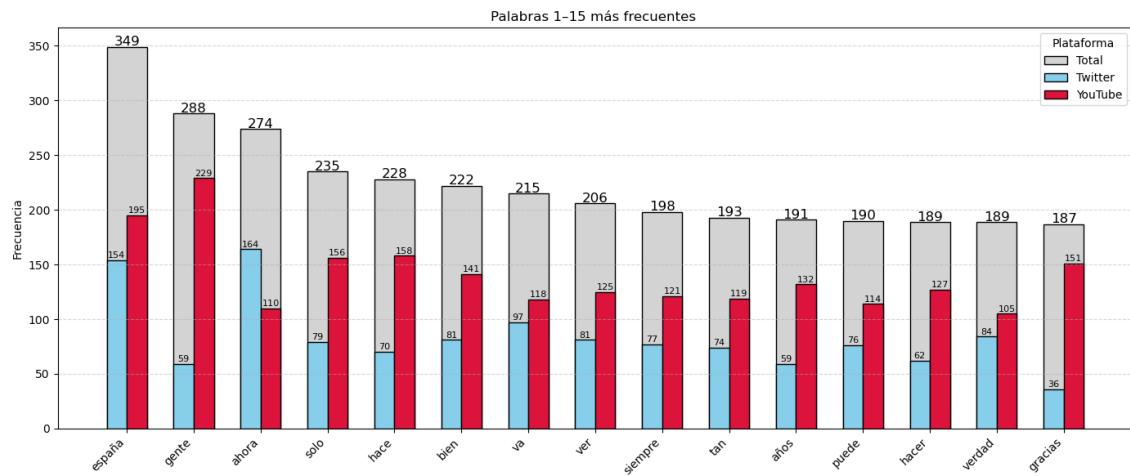


Ilustración 13: gráfica de barras de palabras 1-15 separadas por plataforma

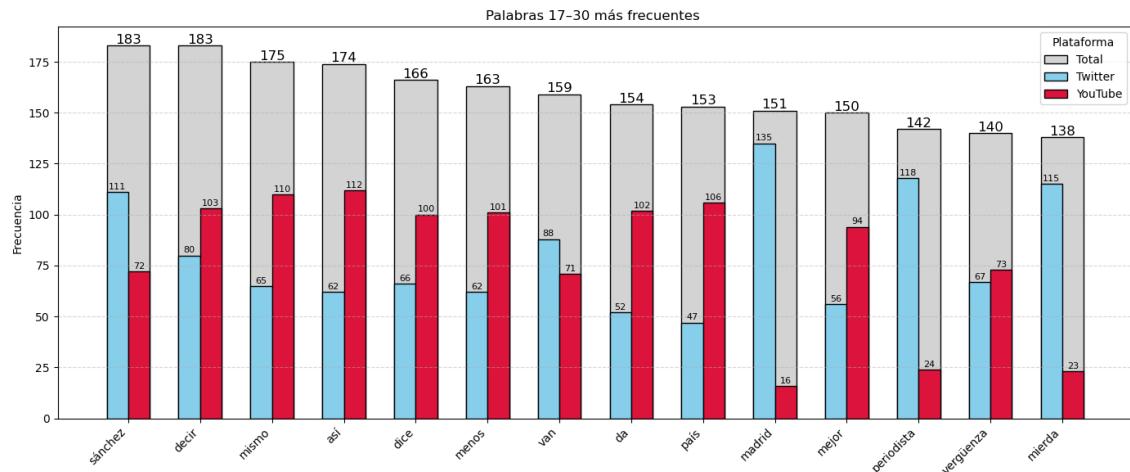


Ilustración 14: gráfica de barras de palabras 16-30 separadas por plataforma

Se puede ver la palabra más usada es “españía” aparece 349 veces mientras que la segunda (que es “gente”) aparece 288 veces.

Si se compraren las palabras más frecuentes y se comparan con el wordcloud que mostraba todas las palabras de vocabulario (“Ilustración 11: wordcloud de todas las palabras”) se puede ver que “españía” sigue siendo es la más usada y no tiene un incremento sustancial por plataforma, la palabra “periodista” se encuentra en la posición 28 teniendo claramente mucha relevancia en Twitter y las palabras “mujer” y “hombre” no aparecen en las 30 palabras más usadas.

También se puede ver que la palabra “Sánchez” aparece más veces en Twitter que en Youtube, pero no es una diferencia que denote su uso en una plataforma de manera principal. Si, en cambio, se observa en la palabra “Madrid” se puede ver que su uso cambia dependiendo de la plataforma, teniendo dominancia en Twitter.

Si se hace el mismo estudio, pero usando los comentarios que se han identificado como que contienen odio se consigue el siguiente wordcloud:



Ilustración 15: wordcloud de comentarios con odio por plataforma

Se puede ver qué en dichas condiciones, las palabras “España”, “gente”, “hace” y “periodista” siguen siendo muy importantes.

Este gráfico aporta mucha información sobre la diferenciación entre las plataformas, se puede ver que, en este wordcloud, las palabras como “España”, “periodista” y “mierda” siguen siendo de las más grandes, pero solamente en la plataforma de Twitter, en Youtube pierde importancia.

Para poder analizar más fácilmente la cantidad de palabras en función de la plataforma se volverán a usar los gráficos de barras de la página anterior.

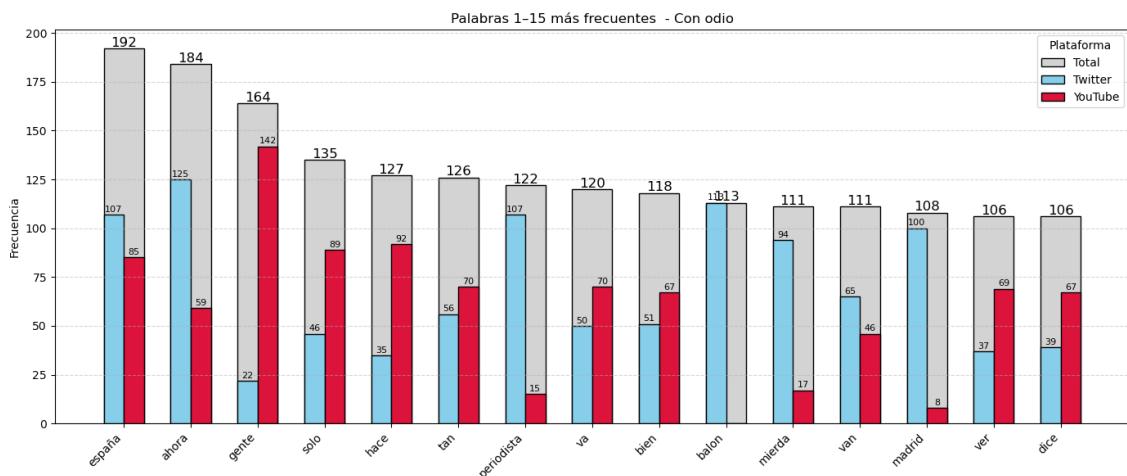


Ilustración 16: gráfica de barras de odio de palabras 1-15 separadas por plataforma

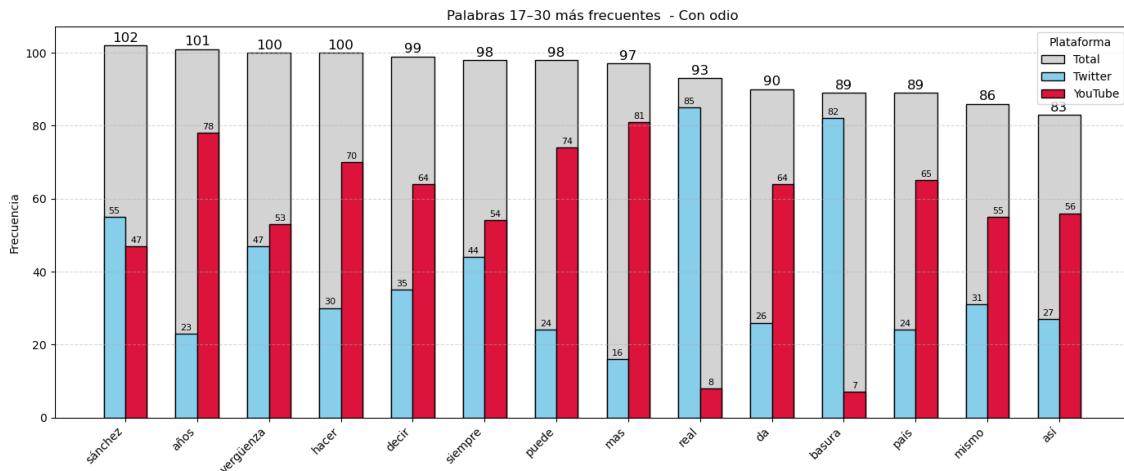


Ilustración 17: gráfica de barras de odio de palabras 16-30 separadas por plataforma

Se puede ver como la palabra más usada sigue siendo “España”, usándose más en Twitter que en Youtube (al revés que en la gráfica que contiene todos los menajes). También se puede ver que palabras como “periodista” pasan de la posición 28 a la 7 cuando se tiene en cuenta el odio.

Viendo la diferencia que hay entre la cantidad de palabras que hay entre el total y la clasificación de odio se puede intuir que hay una moderación previa a la recolección de estos datos que afecta a los datos recibidos. Esto hace que algunas palabras aparezcan más veces en una plataforma que en otra, ya que los usuarios se adaptan a la moderación de cada plataforma.

La diferencia vista en las gráficas es suficiente como para confirmar que los usuarios se comportan distinto entre plataformas, pero aporta información que puede ser útil para más adelante.

Si se repite el estudio, pero, esta vez con solamente los comentarios que contienen odio se puede ver el siguiente wordcloud:



*Ilustración 18: wordcloud de comentarios sin odio por plataforma*

Se puede ver como las palabras más usadas son “españa”, “gracias”, “gente”, “siempre” y “hace”.

También se puede ver que las palabras del entorno más político como “pp”, “Sánchez” o “ayuso” aparecen más veces que en el wordcloud de odio.

Si miramos las cantidades de palabras de los textos sin odio, se puede ver lo siguiente:

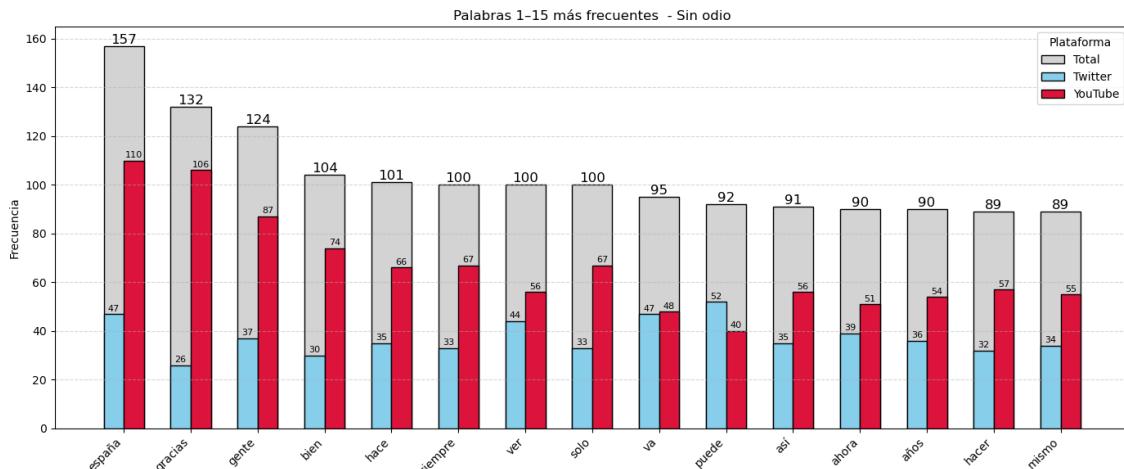


Ilustración 19: gráfica de barras sin odio de palabras 1-15 separadas por plataforma

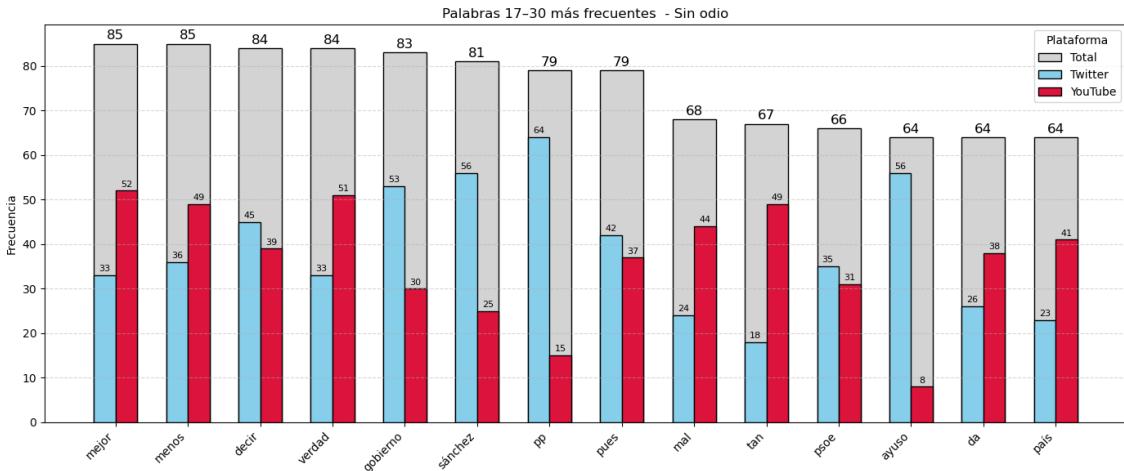


Ilustración 20: gráfica de barras sin odio de palabras 16-30 separadas por plataforma

Se puede ver que la palabra más usada sigue siendo “españa”, pero esta vez predomina en la plataforma de Youtube, en el estudio de las palabras con odio se ha visto que “españa” aparece principalmente en Twitter. Esto puede deberse a que en Twitter es una plataforma mucho más política que Youtube.

También se puede ver que la palabra “periodista” ya no forma parte de las 30 palabras más usadas (en la clasificación de odio aparecía como la séptima palabra). A las palabras “vergüenza” y “basura” les pasa lo mismo, desaparecen en el contexto de sin odio.

Las palabras como “pp”, “psOE”, “ayuso” y “gobierno” no aparecían en el contexto con odio, pero ahora sí. Las palabras relacionadas con el PP (es decir “pp” y “ayuso”) predominan en la plataforma de Twitter, mientras las otras (“psOE” y “gobierno”) aparecen por igual en las dos plataformas.

## 6.2.2 Estudio de los emojis

Los emojis son elementos del texto que aportan información no textual al mensaje, añadiendo información sobre el estado de ánimo del emisor o el sentimiento del texto. Como representan elementos textuales tradicionales no se pueden tratar de la misma manera que las palabras, por eso se ha hecho un estudio aparte para analizarlos.

Al igual que en el apartado “Estudio de los textos” se ha utilizado dos graficas para hacer el estudio principal de los emojis, primero se han utilizado emojiclouds (versión de los wordclouds, pero para emojis) y después se han utilizado gráficos de barras para poder cuantificar su presencia en el vocabulario.

Antes de mostrar un emojicloud tradicional de todos los emojis, se va a visualizar un emojicloud que no asocie tamaño a los emojis en función de su frecuencia, de este modo se puede ver todo el vocabulario:



Ilustración 21: emojicloud de todos los emojis

Se puede ver que hay muchos emojis en el vocabulario de las palabras. Para poder extraer alguna información se va a representar los emojis que aparezcan más de 500 veces, el resultado se puede ver en la siguiente ilustración:



Ilustración 22: emoji cloud con los emojis que aparecen más de 500 veces

Como se puede ver en la ilustración anterior, los emojis más utilizados son y . Esto realmente no aporta ninguna información sobre si el vocabulario de los comentarios ya que se pueden utilizar tanto para mostrar odio como no. En cambio, se pude ver emojis como o que sí que tienen una intencionalidad más clara.

Para poder hacer una distinción más clara en un wordcloud se eliminarán los emojis de risa y se mostrarán todos los que aparezcan más de 200 veces (de este modo aparecerán más emojis que en la imagen anterior). El resultado es el siguiente:



Ilustración 23: emoji cloud con los emojis que aparecen más de 200 veces

Si se mira el emoji cloud se puede ver que hay emoticonos que son claramente neutros, como: , , o . También se puede ver que hay otros que aparentan tener más intencionalidad como: , o .

También se puede ver que hay 3 emojis que son cuadrados de color piel. A simple vista parece extraño que la gente utilice este emoticono de un cuadrado. Investigando se ha

descubierto que en realidad la gente no lo está utilizando conscientemente. Estos emoticonos están ocultos al usuario de manera directa, pero al combinarlos con ciertos emojis se combinan.



Ilustración 24: suma de emojis

Esto aporta información importante para poder evaluar el contexto del estudio ya que, suponiendo que la gente utiliza su color de piel para sus emojis, se puede hacer un estudio teniendo en cuenta la etnia de la gente.

Como estos emoticonos de piel no aportan información relevante en el contexto de este estudio, se van a eliminar para poder visualizar un emojicloud final que contenga los emojis que



Ilustración 25: emojicloud con los emojis que aparecen más de 200 veces sin etnia

En el año 2015 se publicó un estudio llamado *Sentiment of Emojis*<sup>16</sup>. En el estudio se investigó el sentimiento que aporta cada emoji. Para realizar el estudio involucraron a 83 etiquetadores para que clasificasen los emojis de 1.6 millones de tweets de 13 lenguas europeas. El resultado fue que encontraron que hay prácticamente consenso entre el uso de los emojis en las 13 lenguas europeas, que la mayoría de los emojis clasificados se consideraban como positivos o neutros y consiguieron asignar una puntuación a 751 emojis.

En 2023, el usuario de github *omar-foss* creó una librería de Python llamada *emosent*<sup>17</sup> que permite conseguir la puntuación de los emojis del estudio *Sentiment of Emojis*.

<sup>16</sup> (Peta Kralj Novak, 2015)

<sup>17</sup> (omkar-foss, 2023)

Gracias al estudio y a la librería se ha podido investigar los emojis más utilizados en función de la puntuación. Las siguientes imágenes son wordclouds generados en función de la puntuación (asociando a sentimiento positivo la puntuación positiva, neutro a puntuación igual a cero y sentimiento negativo a la puntuación negativa).

Los emojis más usados que transmiten sentimiento positivo son los siguientes:



Ilustración 26: emojicloud de emojis positivos

Los emojis más usados que transmiten sentimiento negativo son:



Ilustración 27: emojicloud de emojis negativos

Finalmente, si se miran los emojis neutros, se puede ver:

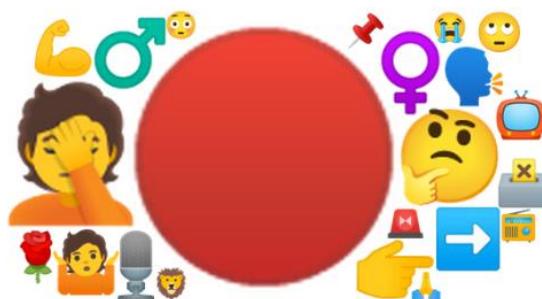


Ilustración 28: emojicloud de emojis neutros

Poder clasificar los emoticonos en función del sentimiento se va a usar la puntuación asociada en el estudio *Sentiment of emojis* que se ha comentado previamente. Para los emojis que no tienen una puntuación asociada (fueron creados después del estudio) se les ha asignado una puntuación de manera manual.

Para poder visualizar de manera numérica que emojis tienen más importancia se va a hacer un gráfico de barras como el del apartado anterior (el apartado “Estudio de los textos”).

El resultado de las gráficas de barras es el siguiente:

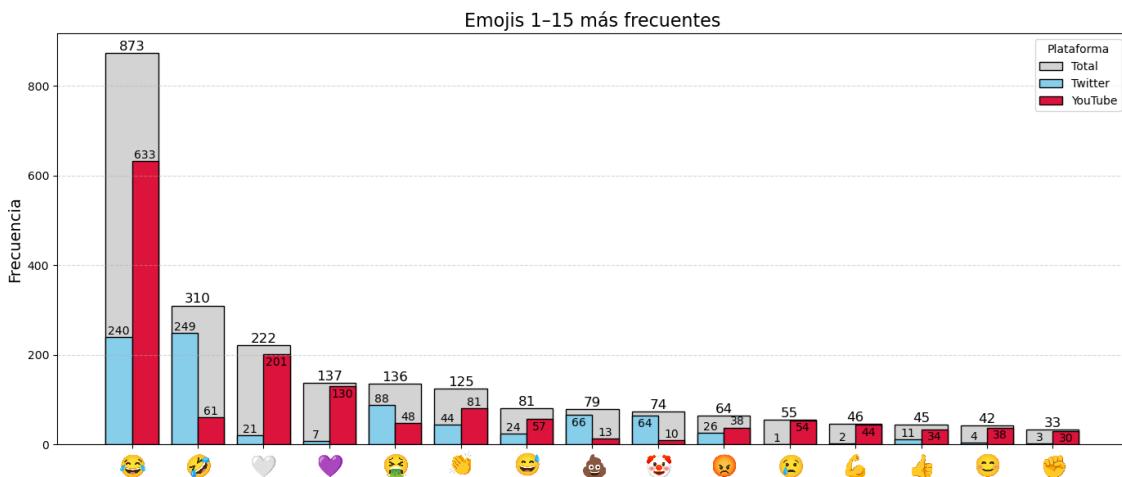


Ilustración 29: gráfica de barras de emojis 1-15 separadas por plataforma

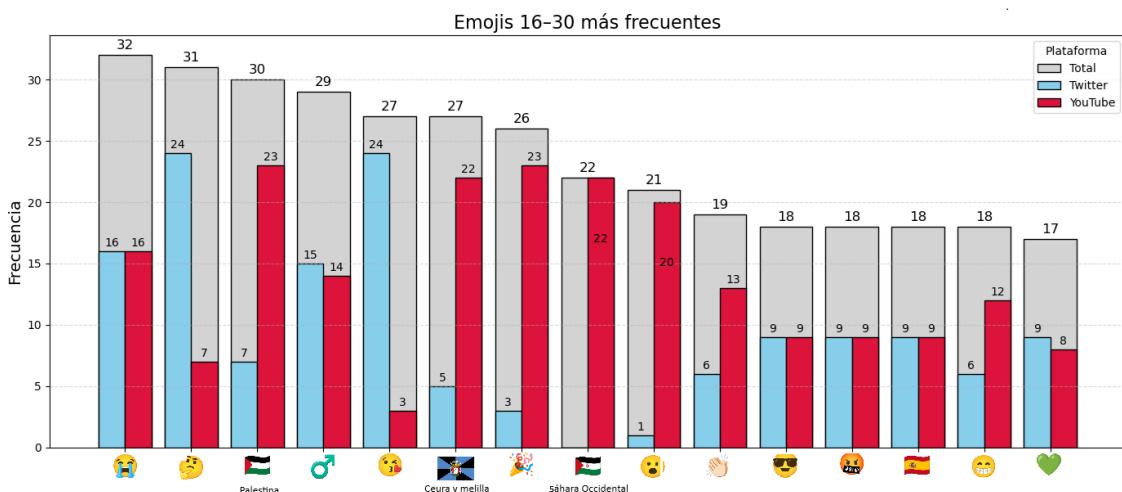


Ilustración 30: gráfica de barras de emojis 16-30 separadas por plataforma

Se puede ver que los emojis más utilizados son los 😂 (con 873 usos) y 🤣 (con 310 usos), se puede ver que 😂 se usa principalmente en Youtube y 🤣 se usa principalmente en Twitter.

Los emoticonos relacionados con corazones o banderas se utilizan principalmente en Youtube (con excepción con la bandera española y el corazón verde) mientras que los emojis que se utilizan principalmente en Twitter no parecen seguir ningún patrón concreto.

Para poder apreciar mejor la proporción de los emojis más usados, se representaron en un gráfico de treemap. Este gráfico divide en tantas secciones como tipos elementos se han de mostrar (en este caso 20), el tamaño de la sección depende de la cantidad de elementos individuales que hay (por ejemplo, hay 2.471 emojis individuales al elegir los 20 tipos más usados, el emoji 😂 aparece 873 veces, por eso ocupa el 35% del espacio total).

Se puede ver que cada celda tiene 3 textos. El que está entre paréntesis representa el total de veces que aparece ese emoji, el texto que empieza por “T:” y numero representa la cantidad de veces que aparece ese emoticono en Twitter y el texto que empieza por “Y:” y un número representa la cantidad de veces que aparece ese emoji en textos de Youtube.

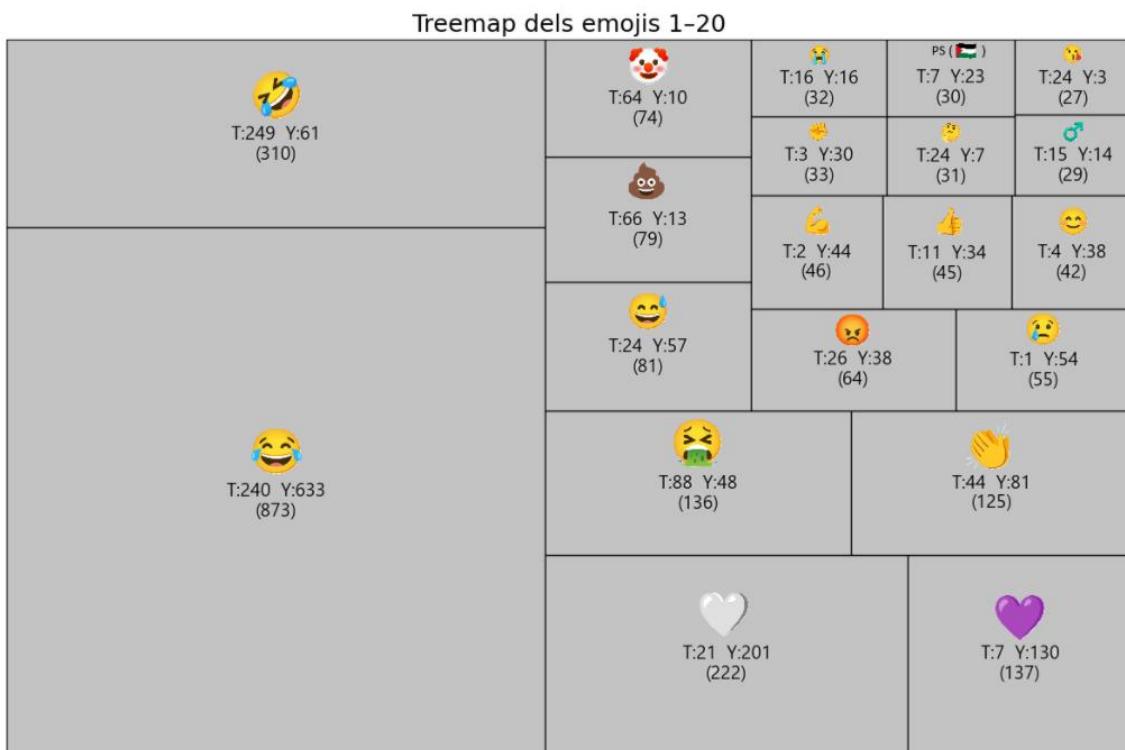


Ilustración 31: treemap de los emojis más usados

Se puede ver que los emojis 😂 y 🤣 representan casi el 50% del total de los emojis (concretamente el 47.8%). Esto puede ser un problema para el uso de emojis como método de clasificación ya que, como se ha mencionado anteriormente, los emojis de risa pueden usarse tanto como para indicar positividad como negatividad, haciendo que potencialmente, el 50% de los emojis, no puedan aportar información adicional.

Afortunadamente, se observa los otros emojis se puede ver que la mayoría aparecen en la clasificación de emojis por sentimiento como positivos o como negativos, haciendo que el otro 50% restante de los emojis del top 20 sí que puedan ayudar aportar información adicional para la clasificación.

A continuación, se va a hacer un estudio con los emojis que se encuentran en comentarios que contienen odio.

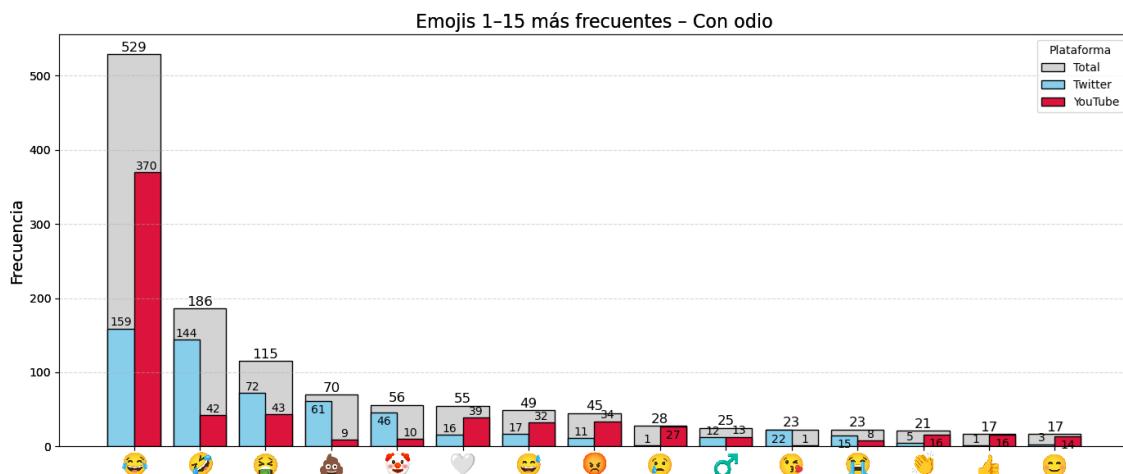


Ilustración 32: gráfica de barras de emojis con odio 1-15 separadas por plataforma

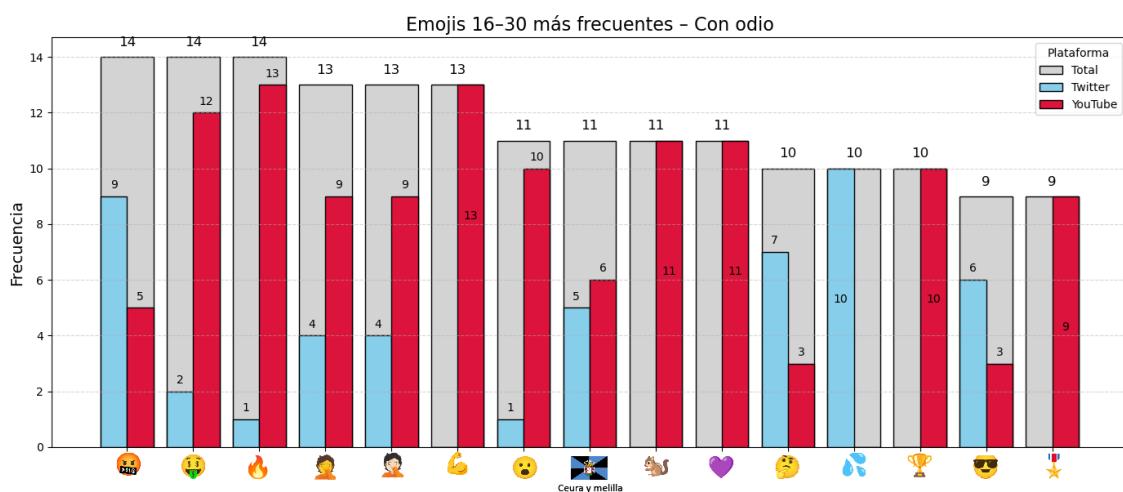


Ilustración 33: gráfica de barras de emojis con odio 16-30 separadas por plataforma

Se puede que los emojis más utilizados son 🤣 y 😢, confirmando que se utilizan tanto como en un sentido positivo como negativo. Si se observan los otros emoticonos en detalle se puede ver que hay algunos, como 🐿️, 💜, la bandera de melilla o ♂ que parecen que están mal clasificados.

Para buscar la explicación de porque estos emoticonos, que aparentemente no tienen significado negativo, se encuentran en contexto de odio se ha creado la siguiente tabla:

	Gracias Begoña por robarnos a todos 😡. Se le acabó la fiesta YA! a todos estos corruptos. Vamos 🐦🐦🐦🐦🐦🐦🐦🐦🐦🐦🐦
---	---

La ardilla es el logo del partido “Se acabó la fiesta” el cuál se presentaba a las elecciones europeas. También se utiliza en otros comentarios para hacer referencia a una rata en modo de insulto.

	@Fgarea Todo el día soltando bulos y mentiras, que ellos saben que se las traga su electorado sin rechistar, debido a su indigencia mental, como dogma de fe.
	GRACIAS CANAL RED !!! VUESTRA VALENTIA Y HONESTUDAD ES TODO UN LUJO EN ESTE PAIS DE CORRUPTELA INSTITUCIONAL. ADELANTE PODEMOS 🎉🎉🎉🎉❤️❤️❤️❤️❤️❤️❤️❤️❤️❤️

EA	Me encantan las entrevistas de Rescue you, ahí se ve muy claro la sociedad que tenemos, el poco cerebro de la sociedad, perversión total adoctrinado por los medios de comunicación pagados con dinero público Un crack Rescue you 🙄 EA👍
	@iescolar ""Gañán"" sigue mintiendo y falseando las informaciones a tú manera que ya no te cree ni tú propia madre ""mamarracho"" EAEAEA

En muchos dispositivos, el símbolo de la bandera de melilla no está disponible, haciendo que en muchos dispositivos se muestre como “EA”. Los tweets que lo utilizan lo hacen como expresión, no como bandera.

👉: este símbolo se está clasificando como odio ya que hay emoticonos que lo utilizan como “suma” (explicación de “Ilustración 24: suma de emojis”). Haciendo se cuente 👉 cuando en verdad los comentarios utilizan emoticonos como 🙄 en modo de burla, no utilizan el emoticono de manera directa.

También se puede ver que el uso de los emojis varía dependiendo de la plataforma, no hay muchos emoticonos que se utilicen por igual en ambas plataformas. Esto podría deberse a que todas tienen sistemas de moderación de los comentarios, siendo posible que los usuarios usen unos emojis u otros en función de lo que permita la plataforma. En caso de que se confirmara dicha hipótesis, no se podrían utilizar los dos tipos de datos conjuntamente ya que los usuarios se comportan de distinta manera, limitando la cantidad de datos que se pueden utilizar en los modelos.

Para comprobar dicha hipótesis se creó un modelo de clasificación de plataforma. Con este modelo se comprobará si hay algún método para separar los datos, es decir, para distinguir si las plataformas tienen comportamientos distintos. El modelo se encuentra en el capítulo “Clasificación de plataforma”.

A continuación, se va a investigar la diferencia entre los emojis en los comentarios positivos:

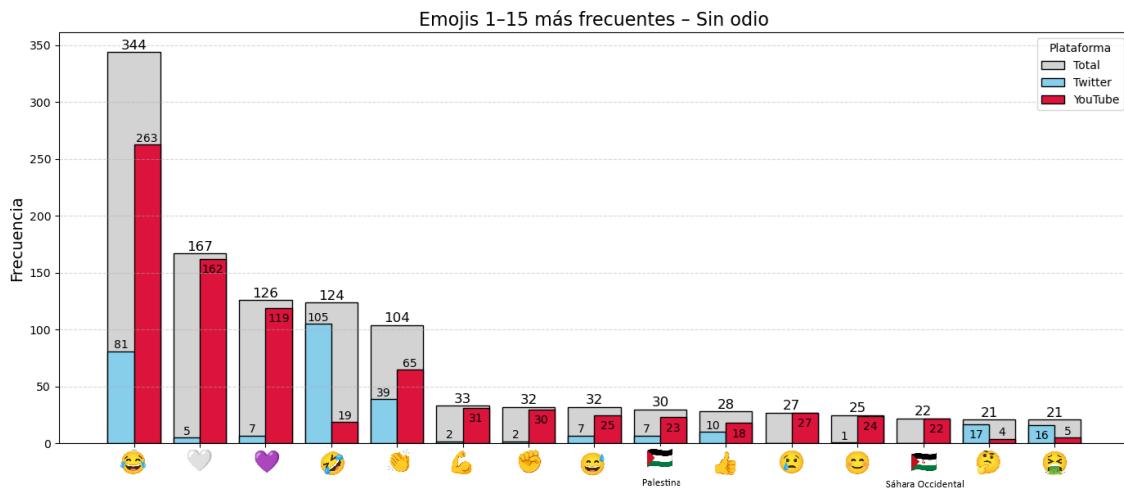


Ilustración 34: gráfica de barras de emojis sin odio 1-15 separadas por plataforma

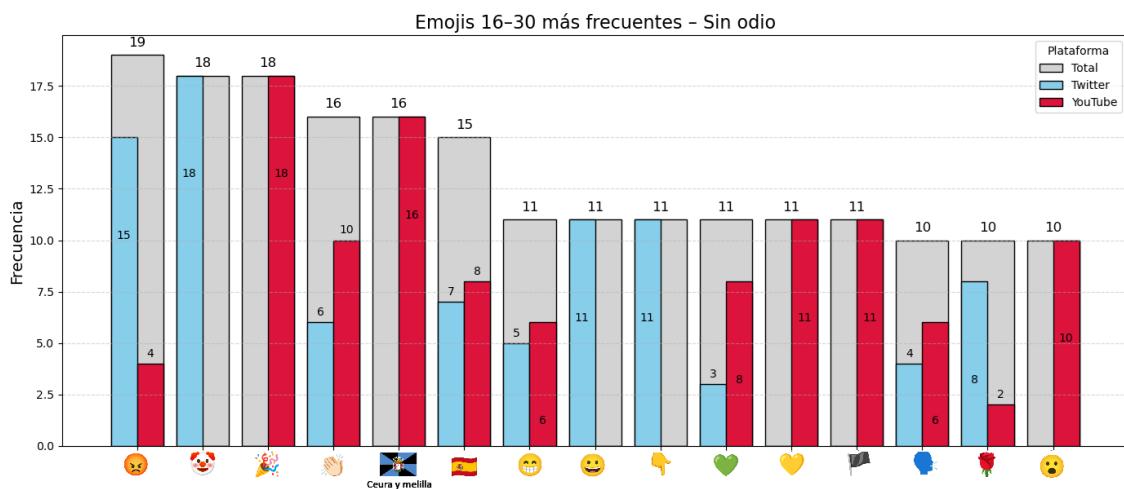


Ilustración 35: gráfica de barras de emojis sin odio 16-30 separadas por plataforma

Se puede ver que los emoticonos de risa y de corazones blancos y morado son los más utilizados. A continuación, vienen emoticonos que según el estudio de “Sentiment of Emojis” se consideran como neutros.

También se puede ver que la cantidad de veces que se utiliza el emoticono de risa es mucho superior en la categoría con odio, aun así, se sigue utilizando mucho más en la plataforma de Youtube que en Twitter.

## 7 Dataset

Blanquerna tiene una lista de periodistas que suben videos de Youtube. Se ha pedido que se recolecten los comentarios y la transcripción de los videos del período de las elecciones del Parlamento Europeo del año 2024.

Para poder recolectar estos datos se han desarrollado un script que consulta a la API de Youtube y consigue los datos de cada uno de los videos de los periodistas que se han pedido.

### 7.1 Script Youtube

Google Cloud tiene acceso a una API de Youtube<sup>18</sup> que permite a los desarrolladores hacer consultas para automatizar múltiples acciones, como: consultar videos, subir videos, buscar comentarios, contestar a comentarios, borrar comentarios, eliminar videos, editar información de videos o del canal, etc.

Cada usuario tiene acceso a **10.000 créditos al día** para hacer consultas. Cada consulta consume un número diferente de créditos. Las consultas que se tienen que hacer son las siguientes:

1. Consultar id canal (acción “*search list*”).
2. Consultar id videos del canal en el período de tiempo necesario (acción “*search list*”).
3. Consultar comentarios del video (acción “*comments list*”).

Cada acción tiene un coste de cuota distinto. Las cuotas de consumo son las siguientes<sup>19</sup>:

1. Acciones “*search list*”:

search	list	<b>100</b>
--------	------	------------

2. Acciones “*comments list*”:

comments	list	<b>1</b>
	insert	<b>50</b>
	update	<b>50</b>
	setModerationStatus	<b>50</b>
	borrar	<b>50</b>

---

<sup>18</sup> (Youtube, 2025)

<sup>19</sup> (Google Cloud, 2025)

La lista de periodistas proporcionada por Blanquerna es de 29 periodistas de los cuales 19 tienen videos en Youtube.

Para cada periodista se van a hacer las consultas mencionadas previamente, es decir, el coste de créditos por periodista es de:

1. Consultar id canal = 100 créditos
2. Consultar id videos de máximo 100 videos de un canal en el período de tiempo necesario = 100 créditos
3. Consultar máximo 100 comentarios de un video = 1 crédito

El coste total de la búsqueda de comentarios en Youtube es de:

$$n^o \text{ periodistas} * (\text{coste consulta id canal} + \text{coste consulta id videos} + \text{coste consulta comentarios} * n^o \text{ videos})$$

Si se analiza la fórmula anterior se puede ver que es muy importante tener en cuenta el orden de las consultas al hacer el script. Si asumimos que no se consultan comentarios (para simplificar el cálculo) se puede que cada ejecución consume:

$$19 \text{ periodistas} * (100 + 100 + 0) = 3.800 \text{ créditos} = 38\% \text{ de la cuota diaria}$$

La estrategia de consulta tiene que hacerse en dos fases, la primera para encontrar los videos que se van a consultar y la segunda fase para consultar los videos. De este modo solo se tiene que consultar los videos una vez y así se puede maximizar el número de consultas de comentarios que se hacen a la api.

Este método reduce el consumo drásticamente, consultando una vez los videos y guardándolos en un fichero se evita el consumo de 3.800 créditos por ejecución, reduciendo el total a potencialmente:

$$10.000 \text{ créditos} - 3.800 \text{ créditos} = 6.200 \text{ créditos}$$

$$6.200 \text{ créditos} = 6.200 \text{ consultas} * 100 \text{ comentarios/consulta} = 620.000 \text{ comentarios}$$

Si se tienen que buscar los videos se pueden conseguir 620.000 comentarios en un día, si la información de los videos ya se ha seleccionado se puede llegar a aumentar a potencialmente 1.000.000 de comentarios por día.

## 7.2 Transformaciones de textos / limpieza

Antes de poder usar los textos para entrenar modelos hace falta procesarlos. Hay elementos de frases que pueden empeorar la interpretación de las palabras y se tienen que transformar o eliminar para que no interfieran en la interpretación de los modelos.

El texto que tiene un tweet está compuesto por varias partes, que son:

- Identificador de retweet: Si un tweet es la respuesta a otro tweet, el mensaje de texto empieza por “RT”.
- Menciones: cuando un mensaje hace referencia a otro usuario se usa un elemento que tiene el formato “@<nombre\_usuario>”.
- Texto: es el mensaje que transmite el usuario.

A continuación, se muestran unos tweets de ejemplo:

1. '@iescolar Al menos no recibe felicitaciones y agradecimientos de golpistas y terroristas.'
2. 'RT @palomarando: Letizia en Vetusta y yo en la presentación del libro de mi amigo @martinbianchi. <https://t.co/JFpNfRVbyZ>'
3. 'RT @ElSaltoDiario: 💰 Un tasa de tan sólo el 2% de sus riquezas podría llegar a recaudar entre entre 200.000 y 250.000 millones de dólares de...'

Adicionalmente a los retweets y menciones, el texto del tweet, también se tiene que procesar. Los cambios que se van a hacer son:

- Eliminar URL
- Eliminar caracteres de alfabetos que no sean el romano.
- Eliminar números.
- Eliminar símbolos como “.”, “,”, “(“, “{“, etc.
- Eliminar espacios múltiples
- Eliminar espacios al final y al principio de las frases
- Transformar los emojis a *tokens* (este cambio se hace en función del estudio, hay pruebas que tienen los emojis tokenizados y otros que los separan para procesarlos luego).

Al hacer estas modificaciones, los mensajes, acaban siendo así:

1. ‘al menos no recibe felicitaciones agradecimientos de golpistas terroristas’
2. ‘letizia en vetusta y yo en la presentación del libro de mi amigo’
3. ‘money\_bag un tasa de tan sólo el de sus riquezas podría llegar recaudar entre entre millones de dólares de’

Una vez se ha limpiado el texto se tienen que codificar las palabras para que los modelos de Machine Learning y Deep Learning puedan interpretarlas.

## 7.3 Encodings y embeddings

Los modelos de *Machine Learning* (ML) son algoritmos que encuentran conclusiones de datos mediante la aplicación de fórmulas matemáticas por lo que no siempre pueden interpretar los datos directamente. Si el problema que se pretende solucionar consiste en generar diagnósticos de datos médicos o en analizar métricas de un coche de fórmula 1 el modelo puede entender las medidas de los sensores ya que normalmente se representan con valores numéricos, si en cambio lo que se pretende es identificar elementos en imágenes o buscar patrones en textos (como es el caso) se tienen que transformar los datos a valores numéricos para que así el modelo pueda extraer patrones y encontrar resultados.

Para poder convertir los textos en números se han creado distintos métodos de codificación de textos, unos son más complejos y otros menos, la diferencia recae en el coste computacional y la pérdida de información que tienen dichos métodos.

Los métodos que se explicarán en este apartado van de más sencillo a más complejo, pudiendo entender y apreciar la necesidad de utilizar una representación compleja para las palabras que tenga en cuenta las relaciones entre ellas.

Los métodos más simples requieren de un procesamiento adicional a las palabras, esto se debe a que al ser métodos simples necesitan eliminar elementos que no sean esenciales para funcionar correctamente.

Normalmente se hacen los tres procesados siguientes:

- Stemming: consiste en recortar las palabras para solo dejar la raíz, de esta manera se consigue eliminar la cantidad de palabras. Es un método muy agresivo ya que se juntan palabras con la misma raíz independientemente del contexto.
  - Ejemplo: “Los niños estaban jugando tranquilamente en el parque” → “los niñ esta jug tranquil en el parq”
- Lemmatization: consiste en transformar una palabra en su forma base o lema, de este modo se eliminar el contexto gramática. Es más precisa que stemming.
  - Ejemplo: “Los niños estaban jugando tranquilamente en el parque” → “los niño estar jugar tranquilo en el parque”
- Eliminación de stop-words: consiste en eliminar las palabras no funcionales como artículos, conjugaciones, preposiciones, etc. De este modo solamente quedan las palabras más esenciales en el texto como adjetivos, nombres y verbos.
  - Ejemplo: “Los niños estaban jugando tranquilamente en el parque” → “niños estaban jugando tranquilamente parque”

Estos métodos de procesamiento adicionales solamente se usarán para one-hot encoding y para TF-IDF. Los otros métodos de codificación no los necesita ya que son más avanzados.

### 7.3.1 One-hot encoding

One-hot<sup>20</sup> encoding es una técnica muy básica de codificación de palabras. Consiste en crear un vector de tantos elementos como número total de palabras hay en los textos a analizar donde cada posición representa una palabra del vocabulario, luego se cuenta cuantas veces aparece la palabra en cada unidad textual (en este caso en cada tweet o comentario) y se anota en el vector. El resultado es un valor que tiene el valor 1 o la frecuencia de la palabra en la frase y un 0 cuando no aparece en la frase.

Para apreciar mejor este método se va a ver un ejemplo. Se va a convertir en one-hot encoding los siguientes tweets: “Me gusta el helado”, “No me gusta el calor” y “El helado no me gusta”.

- Vocabulario: me, gusta, el, helado, no, calor
- Resultado one-hot encoding:

*Tabla 1: ejemplo one-hot encoding*

	Me	Gusta	El	Helado	No	Calor
Me gusta el helado	1	1	1	1	0	0
No me gusta el calor	1	1	1	0	1	1
El helado no me gusta	1	1	1	1	1	0

Se puede ver que el resultado es una lista de vectores que representan la cantidad de palabras que hay en el vocabulario.

Se puede ver que esta representación es muy simple, solamente tiene en cuenta la cantidad de veces que aparece una palabra en el vocabulario, eliminando cualquier orden de las palabras.

### 7.3.2 TF-IDF

TF-IDF<sup>21</sup> es una técnica de codificación que se creó con la intención de mejorar el método de one-hot encoding.

El funcionamiento es muy similar, se crea un vector de la dimensión del vocabulario para cada frase y se contabiliza la cantidad de palabras que aparecen por cada frase. La diferencia recae en que esta vez no se guarda un 1 o la frecuencia de la palabra, se guarda el valor de la **Frecuencia del Término (TF)** en el documento multiplicado al valor **Inverso de la Frecuencia en el Documento (IDF)**. Las fórmulas TF y IDF son las siguientes<sup>22</sup>:

---

<sup>20</sup> (Google developers , 2025)

<sup>21</sup> (Google developers, 2025)

<sup>22</sup> (Karabiber, 2025)

*Ecuación 1: fórmula TF*

$$TF = \frac{\text{frecuencia palabra en documento}}{\text{numero de elementos en documento}}$$

*Ecuación 2: fórmula IDF*

$$IDF = \log \left( \frac{\text{cantidad total de documentos}}{\text{numero de documentos en el corpus que contienen la palabra}} \right)$$

*Ecuación 3: fórmula TF-IDF*

$$TF - IDF = TF * IDF$$

Para visualizar mejor el proceso se va a repetir el ejemplo anterior de one-hot encoding (“Tabla 1: ejemplo one-hot encoding”) pero aplicando TF-IDF:

- Vocabulario: me, gusta, el, helado, no, calor
- Resultado one-hot encoding:

*Tabla 2: ejemplo TF-IDF*

	Me	Gusta	El	Helado	No	Calor
Me gusta el helado	0.29	0.41	0.29	0.41	0	0
No me gusta el calor	0.29	0.29	0.29	0	0.41	0.41
El helado no me gusta	0.24	0.32	0.24	0.32	0.32	0

Se puede ver que la codificación TF-IDF tiene el mismo problema que la codificación one-hot encoding con la pérdida de la secuencia de las palabras, pero esta vez se tiene en cuenta la frecuencia de las palabras en el texto.

El resultado es una matriz que contiene un valor que tiene en cuenta la cantidad de elementos que aparecen en cada documento del corpus (es decir cada tweet) y su frecuencia en la que aparecen

### 7.3.3 Word2Vec

En el año 2013 un equipo de investigación en Google publicó el paper “Efficient Estimation of Word Representations in Vector Space”<sup>23</sup>. Este artículo marcó un hito en el campo del Procesamiento del Lenguaje Natural (NLP) al encontrar un método de representación de las palabras en forma de vector representado en un espacio continuo.

Tradicionalmente se usaban métodos como One-Hot encoding o TF-IDF, que representan las palabras de forma muy dispersa, con alta dimensionalidad y que no representan el significado. Este método de representación tiene la ventaja de que codifica las palabras teniendo en cuenta su significado y contexto, agrupándolas en el espacio en función del uso, haciendo que el significado de las palabras se agrupe de manera espacial.

Esto significó una revolución para el campo del NLP ya que se consiguió codificar el significado de las palabras de manera algebraica, de esta manera se puede contextualizar mejor las palabras dentro del vocabulario.

Una de las propiedades fascinantes de este enfoque de representación es que, al tratar las palabras como vectores, se pueden realizar operaciones matemáticas con sentido semántico y encontrar como resultado una palabra.

Si se mira la siguiente imagen se puede ver que la palabra “Man” está cerca de “Woman” y la palabra “King” cerca de la palabra “Queen”, si se hace la siguiente operación: “King” – “Man” + “Woman” da como resultado del vector de “Queen”.

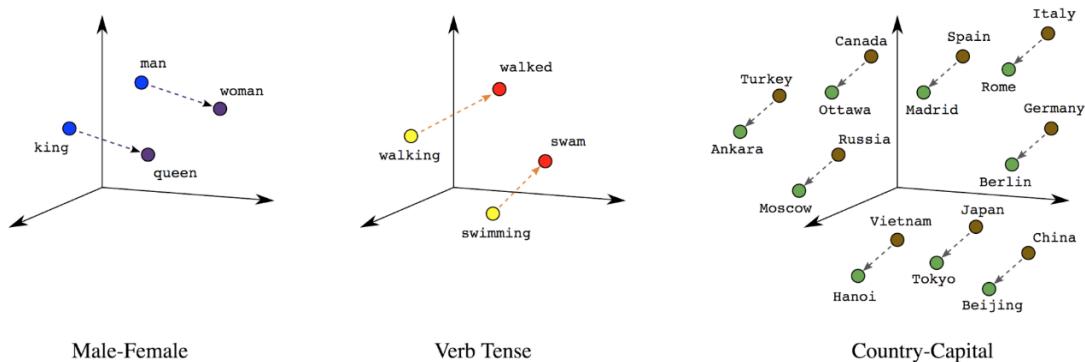


Ilustración 36: representación de las palabras en word2vec

La capacidad de Word2Vec de capturar el significado y contexto de las palabras ha hecho posible que los modelos entiendan las relaciones entre las palabras de una manera que previamente no era posible, dando camino a modelos más sofisticados.

<sup>23</sup> (Tomas Mikolov, 2013)

### 7.3.4 FastText

En el año 2016, un equipo de investigación de Facebook publicó un artículo llamado “Enriching Word Vectors with Subword Information”<sup>24</sup>, donde explicaba una innovadora versión que habían creado de Word2Vec llamada FastText<sup>25</sup>. La implementación<sup>26</sup> que diseñaron tenía la intención de superar algunos de los aspectos de Word2Vec, como por ejemplo con el tratamiento de palabras raras o fuera del vocabulario, es decir, palabras que el modelo no había encontrado durante el entrenamiento.

El enfoque innovador de FastText divide las palabras en n-gramas, al contrario que Word2Vec (que trata las palabras como unidades atómicas), descomponiendo cada palabra en n-gramas (subpalabras) para poder crear un vector conformado por cada parte. Al crear un vector teniendo en cuenta cada parte de la palabra se crea una representación más ajustada al significado real de la palabra.

La siguiente imagen ilustra el proceso de dividir una palabra en trigramas:



Ilustración 37: ejemplo de división de palabras en trigramas<sup>27</sup>

El vector resultante tiene en cuenta cada fragmento (subpalabra), haciendo que las palabras semánticamente similares pero que comparten morfemas o raíces, pero que superficialmente pueden ser distintas, se posicen aún más cerca, mejorando la representación del significado y la capacidad del modelo para inferir el sentido de la palabra que no ha visto durante el entrenamiento.

En resumen, FastText refina el concepto de Word embeddings desarrollado por Word2Vec, mejorando la representación de las palabras en vectores, integrando la información a nivel de grama. Esto crea un sistema de codificación robusto que tiene la capacidad de entender palabras nuevas en el vocabulario ya que al dividirlas puede encontrar su morfología y así encontrar su posición semántica en el espacio vectorial.

<sup>24</sup> (Piotr Bojanowski, 2026)

<sup>25</sup> (Facebook, 2025)

<sup>26</sup> (Facebook, 2025)

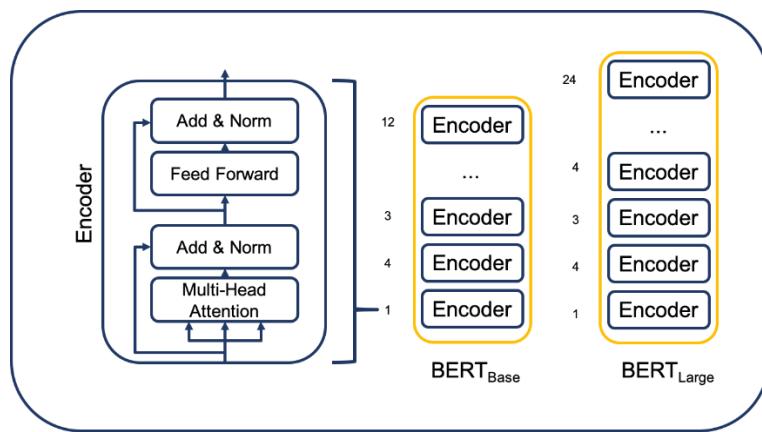
<sup>27</sup> (Chaudhary, 2020)

### 7.3.5 BERT

En el año 2019, Google publicó un artículo que se llama “*BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*”<sup>28</sup>. En este artículo se explicaba como habían creado un *encoder* el cual es capaz de codificar las palabras teniendo en cuenta todo el contexto de la palabra. Esto revolucionó el área del NLP ya que se consiguió una manera de tener en cuenta el contexto tanto de la derecha como de la izquierda simultáneamente, es decir, este modelo es capaz de estar atento al contexto (tiene *self-attention*), algo que los modelos anteriores no podían hacer de forma efectiva y simultánea para ambos lados.

BERT se puede entrenar con datos des de cero o se puede descargar preentrenado (ya contiene datos) y luego ajustar a las necesidades específicas de un problema particular. Esto último se llama *fine-tuning*. Hacer *fine-tuning* es una gran ventaja ya que permite personalizar el modelo de manera que se ajuste al problema que se quiere solucionar.

La estructura interna de BERT es la siguiente:



En la imagen anterior se puede ver que BERT está compuesto por un conjunto *Encoders*, el modelo base está compuesto por 12 y el grande por 24 (BERT large). Cada capa del codificador se compone de múltiples elementos, empieza aplicando *multi-head attention* para entender la importancia de las palabras, luego se normalizan los vectores del *multi-head attention* y se inserta el resultado a una red *feed forward* que realiza la transformación del token y finalmente se vuelve a normalizar. Esto da como resultado una representación del significado de la palabra y la influencia de su contexto.

BERT esta entrenado con textos en inglés, para tener un modelo bueno de clasificación en castellano hace falta utilizar BETO, la versión de BERT preentrenada con textos en español.

<sup>28</sup> (Jacob Devlin, 2019)

## 8 Modelos

Un modelo de Inteligencia Artificial (IA) es un modelo estadístico o algoritmo que encuentra patrones en datos. Puede encontrar patrones en muchos campos distintos como imágenes, texto, audio, datos numéricos o predecir valores futuros, entre otros. Esto permite a los sistemas informáticos de aprender de los datos, extraer información, identificar parámetros y tomar decisiones con la mínima interacción humana. Se considera que es a medianos del siglo XX, con Alan Turing<sup>29</sup>, quien plantea la pregunta: “¿Pueden pensar las máquinas?”.

Para entender el funcionamiento de los modelos hace falta ver los siguientes conceptos:

- Entrenamiento: proceso en que el modelo se expone a los datos para identificar patrones, relaciones o características de los datos. Es la fase en la que el modelo “aprende”.
- Evaluación: es la fase en la que se comprueba si el modelo ha aprendido correctamente o no. Es la fase en la que se “examina” el modelo.
- División de los datos: para hacer que el modelo aprenda bien se deben dividir los datos en dos (una para entrenar y la otra para evaluar) o tres partes (una para entrenar, la otra para evaluar durante el entrenamiento y una para evaluación final). Hacer esto es importante ya que de este modo se puede averiguar si el modelo está memorizando los datos o si está aprendiendo realmente.  
La división de tres partes se hace ya que durante el entrenamiento de una red neuronal se ajustan los parámetros, los parámetros se ajustan, en parte, gracias a los resultados de la evaluación, haciendo que indirectamente los datos evaluados se usen para entrenar. Para tener un resultado final justo se hace una división de evaluación con datos jamás usados, así se puede ver cómo ha aprendido el modelo realmente.
  - Para los modelos de los apartados siguiente se ha hecho una división de 70% de los datos para entrenamiento, el 15% para prueba y el 15% para evaluación final.
- Overfitting: un modelo ha hecho “overfitting” cuando ha memorizado los datos hasta el punto de que no sabe generalizar, ha memorizado hasta el ruido. Un modelo sobreentrenado no es capaz de predecir datos nuevos ya que no ha generalizado el patrón, lo ha memorizado.
- Underfitting: un modelo ha hecho “underfitting” cuando no es capaz de encontrar los patrones debido a su poca complejidad.

---

<sup>29</sup> (Copeland, 2025)

- Matriz de confusión: la matriz de confusión es una matriz que se usa para ver cómo ha clasificado el modelo. En la siguiente imagen se puede ver un ejemplo:

	Actually positive	Actually negative
Predicted positive	TP=40	FP=7
Predicted negative	FN=8	TN=44

Ilustración 39: ejemplo matriz de confusión<sup>30</sup>

Se puede ver que hay cuatro posibles posiciones: True Positive (TP), False Positive (FP), False Negative (FN) y True Negative (TN).

- Métricas de evaluación<sup>31</sup>: para comprobar que el modelo esté aprendiendo correctamente se calcula una métrica para mostrar su rendimiento.
  - Accuracy (exactitud): representa el total de predicciones correctas sobre el total de predicciones.

Ecuación 4: fórmula del accuracy

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Recall (recuperación): es la tasa de verdaderos positivos, la proporción de todos los valores positivos reales que se clasificaron correctamente.

Ecuación 5: fórmula del recall

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

- F1 score: es la proporción de todas las clasificaciones positivas del modelo que realmente son positivas.

Ecuación 6: fórmula de la precisión

$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP}$$

---

<sup>30</sup> (Google, 2025)

<sup>31</sup> (Google, 2025)

## 8.1 Machine learning (ML)

El Machine Learning (ML) o aprendizaje automático es una rama de la inteligencia artificial que ha experimentado una evolución inmensa desde sus inicios. Los primeros modelos de ML consistían en algoritmos sencillos que realizaban regresiones lineales o árboles de decisión. Gracias al avance tecnológico, la capacidad computacional y al incremento de los datos disponibles, el campo ha avanzado exponencialmente, dando lugar a modelos más complejos como las redes neuronales profundas.

En este apartado se van a ver los resultados obtenidos al utilizar tres modelos de ML para la detección de toxicidad. Primero se va a ver de manera teórica su funcionamiento y finalmente se verá qué resultados consiguen al entrenarlos con los distintos métodos de codificación que se han visto en el apartado anterior (“Encodings y embeddings”).

Los modelos elegidos son SVM (Support Vector Machine), RandomForest y, finalmente, XGBoost. La elección de los tres modelos no ha sido aleatoria, se ha realizado para observar cómo clasifica un modelo de ML “normal” (SVM), cómo lo hace uno que utiliza la técnica de Bagging (Random Forest) y luego cómo lo hace uno que utiliza la técnica de Boosting (XGBoost). Ambas técnicas de ensamble se explicarán en el apartado correspondiente de cada modelo.

Para poder hacer el mejor entrenamiento posible se ha utilizado la técnica de optimización Bayesiana para así poder encontrar la mejor configuración de los parámetros.

Los modelos de ML tienen una configuración que define su estructura y comportamiento durante el entrenamiento. La técnica de optimización Bayesiana es un método avanzado que permite encontrar la configuración más óptima de los parámetros de configuración del modelo.

A diferencia de otras técnicas como Grid Search o Random Search que exploran los distintos parámetros de manera exhaustiva o aleatoria, la optimización bayesiana utiliza un modelo probabilístico para modelar la función objetivo (en este caso, la métrica f1). En cada iteración actualiza sus parámetros para calcular dónde es más probable que encuentre el parámetro más óptimo. Esto resulta en una búsqueda más rápida y eficiente que otros métodos de optimización.

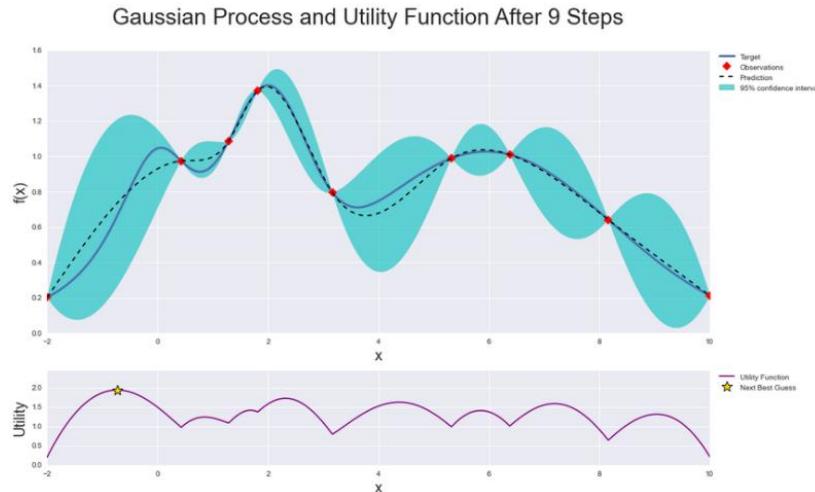


Ilustración 40: proceso de optimización bayesiana<sup>32</sup>

En la imagen se puede ver cómo tras nueve pasos de optimización, se han acotado los posibles valores para optimizar.

Adicionalmente a utilizar los parámetros más óptimos, se ha utilizado cross validation para poder conseguir el entrenamiento más estable posible. Cross validation consiste en dividir el conjunto de datos en  $N$  trozos, luego se usa  $N-1$  fragmentos para entrenar y luego el sobrante para evaluar el entrenamiento. De este modo se puede evaluar si el modelo aprende correctamente y generaliza bien su aprendizaje con los distintos datos o si no. Para el apartado de ML se ha usado un cross validation de cinco fragmentaciones.

La siguiente ilustración muestra como funcionaría cross validation con tres fragmentaciones.

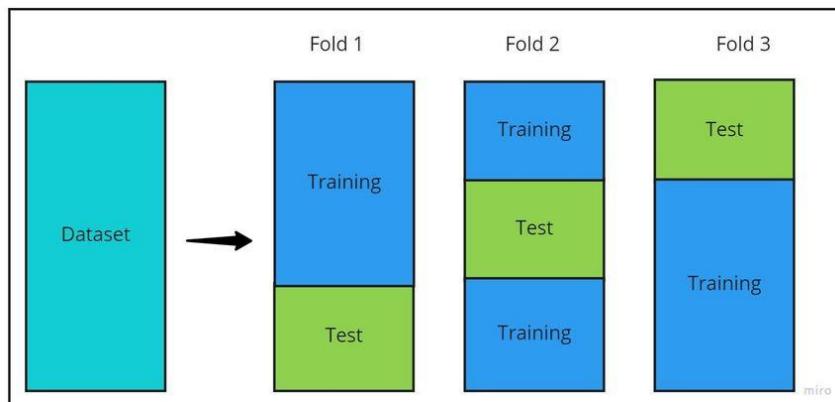


Ilustración 41: ilustración funcionamiento cross validation<sup>33</sup>

<sup>32</sup> (till-m, 2025)

<sup>33</sup> (Lemos, 2022)

### 8.1.1 Support Vector Machine (SVM)

Las **Máquinas de Soporte Vectorial (SVM)**<sup>34</sup> son un algoritmo de aprendizaje supervisado que se utiliza tanto para problemas de clasificación como de regresión, aunque destacan especialmente en las tareas de clasificación. El objetivo principal de un SVM consiste en encontrar un hiperplano óptimo que separe las clases de forma clara en el espacio.

El hiperplano que encuentra SVM separa las clases de cualquier manera, lo hace maximizando el margen, es decir, la distancia entre la frontera y los puntos más cercanos de cada clase, estos puntos se llaman vectores de soporte. A mayor el margen, mejor la clasificación de las clases.

Una de las características de los SVM es su capacidad clasificar datos que no son linealmente separables mediante funciones kernel. Estas funciones permiten a SVM crear dimensiones adicionales para poder separar mejor las clases.

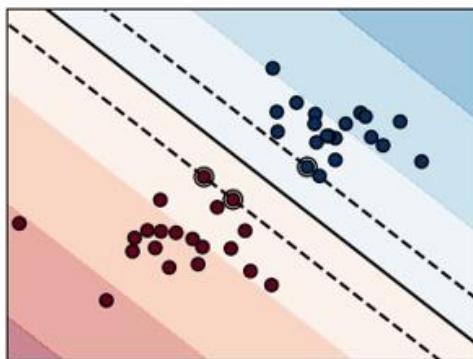


Ilustración 42: representación del funcionamiento de un SVM<sup>35</sup>

Los parámetros que se han utilizado para entrenar el modelo son los siguientes:

- Kernel = “lineal”: define la función del kernel que utiliza el modelo SVM. En este caso se genera un hiperplano lineal.
- C = “1.0”: regula el margen del kernel, evalúa el trade-off de un margen grande y uno pequeño. Esto permite regular la penalización por errores del margen, cuánto más bajo el valor menos errores se penalizan los errores.
- Gamma = “scale”: es el coeficiente del kernel, indica la influencia que tiene cada punto de entrenamiento.

---

<sup>34</sup> (Vapnik, 1995)

<sup>35</sup> (ScikitLearn, 2025)

Los resultados del modelo SVM con la configuración comentada anteriormente son los siguientes:

*Tabla 2: resultados SVM con los distintos embeddings*

Encoding	F1 score	Accuracy	Precision	Recall	ROC-AUC
One-Hot	0.6379 ± 0.1620	0.6051 ± 0.2787	0.6078 ± 0.2875	0.6715 ± 0.2584	0.6429 ± 0.1710
TF-IDF	0.6095 ± 0.0519	0.5633 ± 0.1980	0.5650 ± 0.1929	0.6623 ± 0.2693	0.5438 ± 0.2317
Word2Vec	0.5804 ± 0.3748	0.5553 ± 0.1601	0.5658 ± 0.1013	0.5965 ± 1.0658	0.5539 ± 0.1498
FastText	0.5928 ± 0.1406	0.5420 ± 0.0925	0.5505 ± 0.0908	0.6434 ± 0.9392	0.5380 ± 0.1074
BETO	<b>0.7057 ± 0.1523</b>	<b>0.6950 ± 0.1468</b>	<b>0.7015 ± 0.1730</b>	<b>0.7103 ± 0.3697</b>	<b>0.6945 ± 0.1476</b>

Nota: las varianzas están en escala  $10^{-3}$

Se puede ver que los embeddings de BETO son los que funcionan mejor con el modelo SVM. Esto indica que BETO es capaz de separar los datos de manera espacial, esto hace posible que SVM encuentre un hiperplano que los divide con un valor f1 de 0.7057. Se puede ver que el modelo esta balanceado ya que la precisión (0.7015), *recall* (0.7103) y f1 (0.7057) tienen valores parecidos.

BETO representa las palabras de manera sintáctica y contextual esto enriquece el vector resultante con información que los otros métodos de encoding no representan. Se puede ver que One-hot encoding ha conseguido una puntuación alta en comparación a los otros métodos. Esto debe ser porque muchos de los comentarios de las mismas clases tendrán las mismas palabras, haciendo que los vectores resultantes sean parecidos, por este motivo SVM los separa mejor que otros métodos.

Se puede ver que los métodos de representación que utilizan los vectores semánticos más densos no parecen tener unos resultados especialmente buenos en comparación a métodos más simples.

### 8.1.2 Random Forest

Antes de ver que es un Random Forest, hace falta entender que es un árbol de decisión. Un **árbol de decisión** es un algoritmo de clasificación que divide el espacio en características siguiendo decisiones binarias (por ejemplo: ¿es el objeto rojo? ¿mide el coche más o igual de 200 cm?, etc.).

A continuación, se puede ver una imagen que muestra un ejemplo de árbol de decisión que clasifica a los supervivientes del *Titanic* en función de sus características físicas.

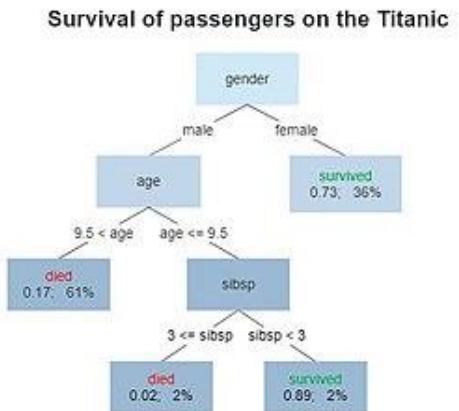


Ilustración 43: ejemplo de árbol de decisión<sup>36</sup>

El árbol se genera hasta que no se puedan generar más divisiones útiles o hasta que se llegue a la profundidad máxima que se le indique al modelo.

Tienen el problema de que tienden a sobreajustar su aprendizaje, pero tiene la ventaja de generar resultados altamente interpretables (ya que se genera un esquema visual de las decisiones que se toman).

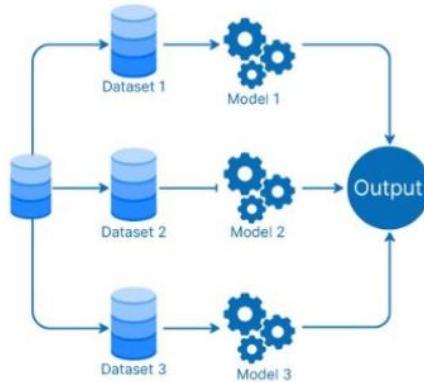
Otro concepto que también hace falta entender antes de ver que es un RandomForest es el de bagging<sup>37</sup>. **Bagging** es una técnica de ensamblado (es decir, combina múltiples modelos “débiles” en uno “fuerte”) que combina modelos de manera paralela. Consiste en generar  $N$  subconjuntos de los datos (llamados bootstraps), luego se generan  $N$  modelos (en este caso árboles de decisión) y se entrena cada uno con cada subconjunto de datos, finalmente se hace una predicción con todos los modelos y se hace un promedio del valor resultante (si el problema es de regresión) o se hace una votación mayoritaria (si el problema es de clasificación).

<sup>36</sup> (Wikipedia, 2025)

<sup>37</sup> (Breiman, Bagging Predictors, 1996)

Este enfoque consigue reducir la varianza del modelo sin incrementar el sesgo, aumentando la estabilidad de este.

A continuación, se puede ver una imagen que representa el funcionamiento de Bagging:



*Ilustración 44: esquema del funcionamiento de bagging<sup>38</sup>*

El modelo RandomForest<sup>39</sup> combina la técnica de Bagging con árboles de decisión y el concepto de aleatoriedad en el proceso de construcción de los árboles. En vez de crear  $N$  árboles y asignar a cada uno con un subconjunto de datos, se asigna a cada uno un subconjunto de los datos, pero con subconjuntos del Bootstrap inicial. Esto hace que los datos de cada árbol tengan una estructura distinta, aumentando la generalización del aprendizaje del modelo y eliminando el problema de que una variable “fuerte” (que divide las clases claramente) que determine siempre el resultado del modelo.

Antes de ver el modelo de Random Forest se va a ver los resultados obtenidos con un árbol de decisión. Los parámetros utilizados para el árbol son los siguientes:

- criterion = “Gini”: criterio que se usa para seleccionar que variable utilizar para hacer la partición de los datos. El criterio de Gini mide como de “pura” es un nodo, es decir, como de homogénea es la clase de las muestras contenidas en él. Se elige la división que minimiza la impureza resultante.
- max\_depth = 10: profundidad máxima del árbol
- min\_samples\_split = 5: número mínimo de muestras que se necesitan para hacer la separación.

---

<sup>38</sup> (Singh, 2023)

<sup>39</sup> (Breiman, Random Forests, 2001)

Tabla 3: resultados DecisionTree con los distintos embeddings

Encoding	F1 score	Accuracy	Precision	Recall	ROC-AUC
One-Hot	0.4701 ± 0.7436	0.6094 ± 0.0950	0.7945 ± 0.0295	0.3345 ± 0.8223	0.6204 ± 0.0809
TF-IDF	0.5096 ± 0.7839	0.6243 ± 0.1533	0.7810 ± 0.1303	0.3788 ± 0.8818	0.6327 ± 0.1403
Word2Vec	0.6154 ± 0.2864	0.5527 ± 0.3412	0.5547 ± 0.4349	0.6945 ± 3.0083	0.5478 ± 0.3991
FastText	0.5724 ± 2.3277	0.5385 ± 0.5598	0.5500 ± 0.4017	0.6050 ± 10.6968	0.5361 ± 0.5656
BETO	<b>0.6376 ± 0.1172</b>	<b>0.6181 ± 0.1513</b>	<b>0.6242 ± 0.1541</b>	<b>0.6517 ± 0.1604</b>	<b>0.6170 ± 0.1547</b>

Nota: las varianzas están en escala  $10^{-3}$

Se puede ver que el árbol de decisión consigue los mejores resultados al usar los embeddings de BETO, esto se debe a que son más complejos y son más ricos en contexto. También se puede ver que la clasificación es equilibrada, no hay ninguna diferencia sustancial entre el *recall*, la precisión y el valor de f1.

A continuación, se va a proceder con el modelo de Random Forest.

Los parámetros que han conseguido el mejor resultado son:

- n\_estimators = 150: es el número de árboles de decisión que se construirán en el bosque. Cuantos más árboles más robustez, pero más coste computacional.
- max\_depth = 50: profundidad máxima de cada árbol, controla la complejidad del modelo.
- min\_samples\_split = 2: número mínimo de muestras que se necesitan para hacer una separación.

Tabla 4: resultados RandomForest con los distintos embeddings

Encoding	F1 score	Accuracy	Precision	Recall	ROC-AUC
One-Hot	0.7110 ± 0.2299	0.7257 ± 0.1075	0.7832 ± 0.6207	0.6527 ± 1.2926	0.7284 ± 0.1099
TF-IDF	0.5825 ± 0.1363	0.5735 ± 0.1782	0.5870 ± 0.1997	0.5782 ± 0.1734	0.5734 ± 0.1831
Word2Vec	0.5918 ± 0.1121	0.5768 ± 0.1464	0.5883 ± 0.1800	0.5956 ± 0.2640	0.5763 ± 0.1533
FastText	0.5881 ± 0.2192	0.5770 ± 0.1304	0.5894 ± 0.1152	0.5871 ± 0.5374	0.5767 ± 0.1274
BETO	<b>0.7448 ± 0.0617</b>	<b>0.7353 ± 0.0655</b>	<b>0.7438 ± 0.1412</b>	<b>0.7462 ± 0.2278</b>	<b>0.7349 ± 0.0682</b>

Nota: las varianzas están en escala  $10^{-3}$

Como era de esperar, todos los resultados obtenidos con el modelo de RandomForest consiguen mejores resultados que el árbol de decisión. Esto refleja la mejora que ofrecen los modelos de *bagging* respecto a los modelos más simples como el árbol de decisión.

En cuanto a los embeddings. BETO sigue consiguiendo los mejores resultados. Esto confirma que la riqueza de la representación sintáctica que consigue es superior a los otros métodos de encoding probados. BETO consigue un valor f1 de 0.7448, las otras métricas como precisión (0.7438) o *recall* (0.7462) están balanceadas, demostrando que el modelo no favorece ninguna clase, el resultado de ROC-AUC (0.7349) confirma que el modelo está aprendiendo a clasificar correctamente.

Se puede ver que One-Hot encoding consigue unos resultados muy parecidos a BERT y muy superiores que otros métodos de codificación de texto. Esto sorprende debido a la simplicidad que tiene One-Hot encoding, no incorpora información semántica ni contextual. Una posible explicación es que las clases a precedir comparten palabras similares en el texto, lo cual favorece que los vectores de One-Hot representen bien las diferencias entre las clases. En cambio, métodos como Word2Vec o FastText no dan tanta importancia a las palabras comunes entre las clases.

### 8.1.3 XGBoost

Antes de poder ver el concepto de XGBoost hace falta saber que es la técnica de Boosting. **Boosting**<sup>40</sup> es una técnica de ensamblado secuencial que combina múltiples modelos “débiles” para hacer un modelo “fuerte”. A diferencia de bagging que lo hace de manera paralela, Boosting entrena los modelos uno a uno, de forma que cada nuevo modelo intenta corregir los errores cometidos por los modelos anteriores. Este método de entrenamiento secuencial mejora los resultados progresivamente.

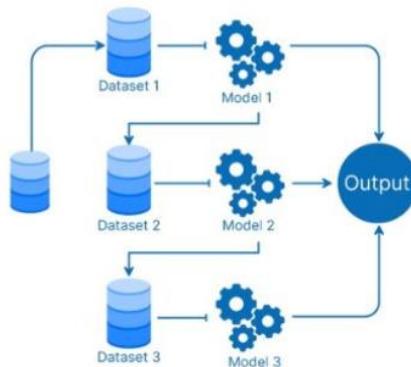


Ilustración 45: esquema del funcionamiento de boosting<sup>41</sup>

<sup>40</sup> (E.Schapire, 1999)

<sup>41</sup> (Singh, 2023)

XGBoost (**eXtreme Gradient Boosting**)<sup>42</sup> es un modelo que aplica la técnica de Boosting a un conjunto de árboles de decisión. Una de sus características más importantes es que limita el crecimiento de los árboles durante la construcción inicial, y posteriormente corta las ramas hasta el umbral de ganancia, evitando de este modo el sobreajuste que tienen los árboles de decisión normalmente.

Este modelo es muy sensible al ruido y se considera malo al gestionar embeddings que tiene vectores densos.

Los parámetros que se han elegido que han conseguido generar el modelo con mejores resultados son:

- n\_estimators = 150: es el número de árboles que se entrenarán secuencialmente.
- max\_depth = 50: profundidad máxima de cada árbol. Durante el proceso de Boosting se deja crecer el árbol “sin control”, luego se corta a max\_depth.
- learning\_rate= 1e-3: tasa de aprendizaje del modelo.
- eval\_metric = “logloss”: métrica usada para evaluar durante el entrenamiento.

A continuación, se puede ver una tabla mostrando el rendimiento de XGBoost en los distintos métodos de embedding:

*Tabla 5: resultados XGBoost con los distintos embeddings*

Encoding	F1 score	Accuracy	Precision	Recall	ROC-AUC
One-Hot	0.6812 ± 0.0315	0.5361 ± 0.4332	0.5295 ± 0.1683	0.9573 ± 1.9209	0.5205 ± 0.5282
TF-IDF	0.6505 ± 0.2321	0.5482 ± 0.2010	0.5403 ± 0.0875	0.8179 ± 1.4232	0.5401 ± 0.2008
Word2Vec	0.6522 ± 0.1478	0.5418 ± 0.2643	0.5356 ± 0.1145	0.8347 ± 0.8975	0.5327 ± 0.2777
FastText	0.6536 ± 0.1375	0.5531 ± 0.1934	0.5439 ± 0.1192	0.8202 ± 1.4328	0.5451 ± 0.2115
BETO	<b>0.6922 ± 0.2487</b>	<b>0.6358 ± 0.2268</b>	<b>0.6155 ± 0.1256</b>	<b>0.7913 ± 1.1016</b>	<b>0.6301 ± 0.2206</b>

Nota: las varianzas están en escala  $10^{-3}$

Los resultados obtenidos con XGBoost muestran que el mejor método de codificación sigue siendo BETO con un valor f1 de 0.6922, pero no supera a los otros modelos por mucho (aproximadamente hay una diferencia de solo 0.05). Se puede ver que en todos los métodos se consigue un valor de recall muy superior a la precisión y f1, indicando que XGBoost tiene preferencia hacia la clase positiva. Esto también se puede ver en el valor ROC-AUC, solo BETO logra un resultado razonable (0.6301), mientras que los otros métodos consiguen valores cercanos al azar, esto sugiere que XGBoost no está discriminando bien las clases.

<sup>42</sup> (Guestrin, 2016)

### 8.1.4 Conclusiones modelos ML

A continuación, se pueden ver tablas comparativas de los resultados conseguidos en cada modelo:

*Tabla 6: comparación f1 final de modelos ML con distintos embeddings*

Encoding	F1 SVM	F1 árbol dec.	F1 Rand.Forest	F1 XGBoost
One-Hot	0.6379 ± 0.1620	0.4701 ± 0.7436	0.7110 ± 0.2299	0.6812 ± 0.0315
TF-IDF	0.6095 ± 0.0519	0.5096 ± 0.7839	0.5825 ± 0.1363	0.6505 ± 0.2321
Word2Vec	0.5804 ± 0.3748	0.6154 ± 0.2864	0.5918 ± 0.1121	0.6522 ± 0.1478
FastText	0.5928 ± 0.1406	0.5724 ± 2.3277	0.5881 ± 0.2192	0.6536 ± 0.1375
BETO	<b>0.7057 ± 0.1523</b>	<b>0.6376 ± 0.1172</b>	<b>0.7448 ± 0.0617</b>	<b>0.6922 ± 0.2487</b>

*Tabla 7: comparación accuracy final de los modelos ML con distintos embeddings*

Encoding	Acc. SVM	Acc. árbol dec.	Acc. Ran.Forest	Acc. XGBoost
One-Hot	0.6051 ± 0.2787	0.6094 ± 0.0950	0.7257 ± 0.1075	0.5361 ± 0.4332
TF-IDF	0.5633 ± 0.1980	0.6243 ± 0.1533	0.5735 ± 0.1782	0.5482 ± 0.2010
Word2Vec	0.5553 ± 0.1601	0.5527 ± 0.3412	0.5768 ± 0.1464	0.5418 ± 0.2643
FastText	0.5420 ± 0.0925	0.5385 ± 0.5598	0.5770 ± 0.1304	0.5531 ± 0.1934
BETO	<b>0.6950 ± 0.1468</b>	<b>0.6181 ± 0.1513</b>	<b>0.7353 ± 0.0655</b>	<b>0.6358 ± 0.2268</b>

*Tabla 8: comparación precision final modelos ML con distintos embeddings*

Encoding	Prec. SVM	Prec. árbol dec.	Prec. Ra.Forest	Prec. XGBoost
One-Hot	0.6078 ± 0.2875	0.7945 ± 0.0295	0.7832 ± 0.6207	0.5295 ± 0.1683
TF-IDF	0.5650 ± 0.1929	0.7810 ± 0.1303	0.5870 ± 0.1997	0.5403 ± 0.0875
Word2Vec	0.5658 ± 0.1013	0.5547 ± 0.4349	0.5883 ± 0.1800	0.5356 ± 0.1145
FastText	0.5505 ± 0.0908	0.5500 ± 0.4017	0.5894 ± 0.1152	0.5439 ± 0.1192
BETO	<b>0.7015 ± 0.1730</b>	<b>0.6242 ± 0.1541</b>	<b>0.7438 ± 0.1412</b>	<b>0.6155 ± 0.1256</b>

*Tabla 9: comparación recall final modelos ML con distintos embeddings*

Encoding	Reca. SVM	Reca. árbol dec.	Reca. Ra.Forest	Reca. XGBoost
One-Hot	0.6715 ± 0.2584	0.3345 ± 0.8223	0.6527 ± 1.2926	0.9573 ± 1.9209
TF-IDF	0.6623 ± 0.2693	0.3788 ± 0.8818	0.5782 ± 0.1734	0.8179 ± 1.4232
Word2Vec	0.5965 ± 1.0658	0.6945 ± 3.0083	0.5956 ± 0.2640	0.8347 ± 0.8975
FastText	0.6434 ± 0.9392	0.6050 ± 10.6968	0.5871 ± 0.5374	0.8202 ± 1.4328
BETO	<b>0.7103 ± 0.3697</b>	<b>0.6517 ± 0.1604</b>	<b>0.7462 ± 0.2278</b>	<b>0.7913 ± 1.1016</b>

Tabla 10: comparación ROC-AUC final modelos ML con distintos embeddings

Encoding	ROC. SVM	ROC. árbol dec.	ROC. Ra.Forest	ROC. XGBoost
One-Hot	$0.6429 \pm 0.1710$	$0.6204 \pm 0.0809$	$0.7284 \pm 0.1099$	$0.5205 \pm 0.5282$
TF-IDF	$0.5438 \pm 0.2317$	$0.6327 \pm 0.1403$	$0.5734 \pm 0.1831$	$0.5401 \pm 0.2008$
Word2Vec	$0.5539 \pm 0.1498$	$0.5478 \pm 0.3991$	$0.5763 \pm 0.1533$	$0.5327 \pm 0.2777$
FastText	$0.5380 \pm 0.1074$	$0.5361 \pm 0.5656$	$0.5767 \pm 0.1274$	$0.5451 \pm 0.2115$
BETO	<b><math>0.6945 \pm 0.1476</math></b>	<b><math>0.6170 \pm 0.1547</math></b>	<b><math>0.7349 \pm 0.0682</math></b>	<b><math>0.6301 \pm 0.2206</math></b>

*Nota:* las varianzas están en escala  $10^{-3}$

Los resultados mostrados en las tablas anteriores confirman que BETO es la representación de texto más potente y versátil para la tarea propuesta.

Esto se debe a que BETO codifica los vectores resultantes teniendo en cuenta el significado semántico y el contexto de las palabras en el texto, capturando de la mejor manera las relaciones entre las palabras de los textos.

Se puede observar que no todos los modelos de ML se benefician por igual con las representaciones de BETO. SVM y RandomForest consiguen aprovechar la forma de representar los embeddings de BETO mientras que XGBoost muestra limitaciones.

Esto demuestra que la elección del modelo y del método de codificación de las palabras es critico para conseguir crear un buen clasificador para la detección de toxicidad en textos.

En conclusión, se puede ver que BETO es una de las mejores alternativas para representar el texto, superando a modelos clásicos, gracias a su capacidad para comprender el contexto y el significado semántico de las palabras.

No obstante, en algunos casos se ha observado que One-hot ofrece resultados sorprendentemente competitivos, especialmente teniendo en cuenta su simplicidad, lo cual sugiere que los patrones léxicos pueden ser capturados sin la necesidad de modelos complejos.

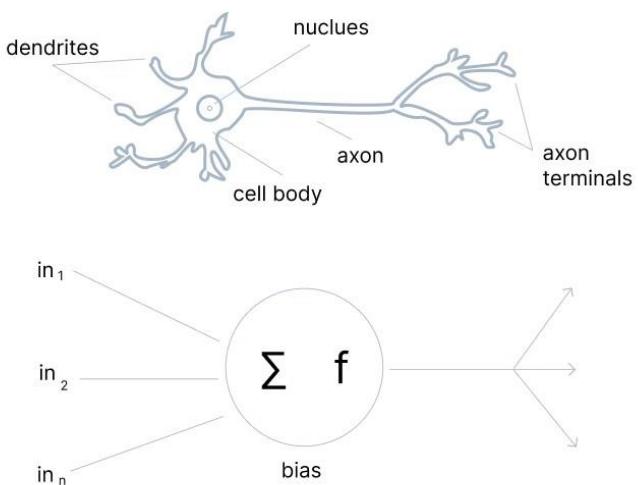
Los experimentos realizados con modelos tradicionales de ML confirman que es posible clasificar los textos en función de su toxicidad.

En el siguiente apartado se explorarán técnicas de *Deep Learning* (DL) con el objetivo de diseñar un modelo que sea capaz de aprovechar aún mejor los embeddings generados por BETO, y mejorar la capacidad de separación de clases, superando el rendimiento conseguido con los modelos SVM, árbol de decisión, Random Forest o XGBoost.

## 8.2 Deep Learning (DL)

El *Deep Learning* es un subcampo del *Machine Learning* en el que se usan redes neuronales artificiales<sup>43</sup> para encontrar patrones complejos en los datos. Su objetivo es imitar (de forma simplificada) el funcionamiento del cerebro humano en cuanto al procesamiento y aprendizaje de los datos.

Al igual que un cerebro biológico, las redes neuronales están compuestas por neuronas. Estas neuronas intentan replicar el funcionamiento de una neurona real, es decir, una neurona recibe un señal o impulso, procesa la información y genera un impulso que se transmite a otras neuronas.



V7 Labs

Ilustración 46: Imagen comparativa entre neurona biológica y artificial<sup>44</sup>

Las neuronas artificiales funcionan de manera muy similar: reciben un valor numérico de entrada (que sería el equivalente al impulso de la neurona biológica), después se hace una operación matemática y finalmente se envía a la siguiente capa de neuronas.

Estas neuronas se organizan en capas, cada capa se conecta entre sí, formando una red de conexiones. Hay capas de entrada (que son las que reciben los inputs); las capas ocultas (*hidden layers*) donde se realizan la mayoría de las operaciones matemáticas y donde la red aprende las relaciones complejas de los datos; y las capas de salida, donde el modelo produce el resultado final.

---

<sup>43</sup> (Google, 2025)

<sup>44</sup> (Baheti, 2021)

En la siguiente ilustración se puede ver un esquema de una red simple:



Ilustración 47: esquema red neuronal simple (imagen propia)

Como los cerebros biológicos, una red neuronal, también tiene que aprender para poder desarrollar su funcionalidad (que puede ser des de clasificar imágenes, a identificar patrones en datos, a predecir valores, etc.).

El proceso de entrenamiento de una red neuronal es fundamentalmente un ciclo de “prueba y error” donde el modelo ajusta los parámetros internos para mejorar las predicciones. Este aprendizaje es iterativo, primero hace *forward pass* y luego *backpropagation*. Cada iteración (también llamado época o epoch) hace los pasos:

1. *Forward pass*: paso donde los datos van des de la neurona de input hasta la de output, cambiando los pesos internos a su paso.
2. Evaluación de los resultados y cálculo del error en las predicciones (loss).
3. *Back propagation*: paso donde se ajustan los pesos de cada neurona a partir del cálculo del gradiente utilizando el valor de la loss del paso 2.

Este método de entrenamiento hace que la red neuronal aprenda de manera mucho más robusta que otros modelos.

Para todos los modelos de red neuronal creados se ha implementado la funcionalidad de *early stopping*. Esta funcionalidad permite a la red tener en cuenta la cantidad de épocas en las que se ha quedado estancada y ya no mejora sus resultados, de este modo puede parar el entrenamiento. Así se ahorra tiempo y consumo de recursos computacionales. Por ejemplo: puede ser que una red neuronal este haciendo un entrenamiento de 10.000 épocas, pero que en la 1.000 ya no esté mejorando, el *early stopping* pararía el entrenamiento antes de llegar a los 10.000 epochs.

### 8.2.1 Multi-Layer Perceptron (MLP)

Un *Multi-Layer Perceptron* es un tipo de red neuronal que fue inventada por Frank Rosenblatt, en el año 1958 publicó un paper llamado “*The perceptron a probabilistic model for information storage and organization in the brain*”<sup>45</sup>.

Un MLP tiene una arquitectura (estructura de neuronas) muy sencilla. Está compuesta, como mínimo, por una capa *input*, una *hidden layer* y un *output*. Normalmente los MLP están compuestos por múltiples capas *hidden*, allí es donde se hacen la mayoría de las transformaciones y donde el modelo encuentra patrones de los datos.

Un diagrama de un MLP podría ser:

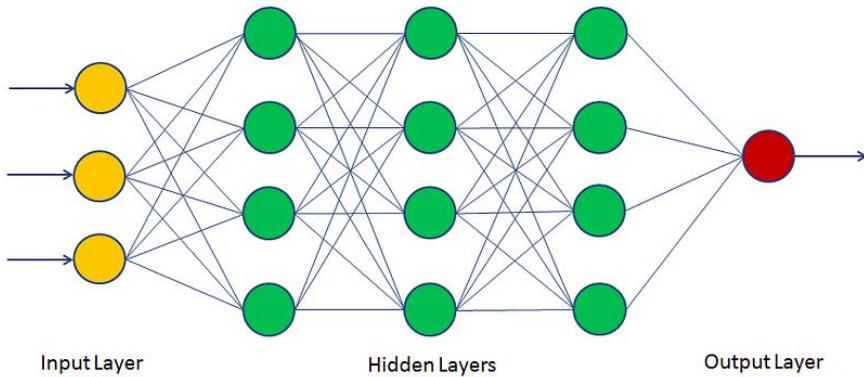


Ilustración 48: Estructura MLP

Para entrenar una red neuronal, el modelo debe ajustar los pesos de sus neuronas a lo largo del tiempo mediante cálculos de forma iterativa. Para hacer el entrenamiento se necesitan tres elementos básicos:

- Función de perdida: mide la diferencia entre la predicción del modelo y la respuesta esperada.
- Optimizador: es el algoritmo que ajusta los pesos del gradiente de la función de pérdida.
- Épocas (*epochs*): es el número de veces que el modelo recorre completamente todo el conjunto de datos de entrenamiento. Normalmente, a mayor el problema más epochs se necesitan.

---

<sup>45</sup> (Rosenblatt, 1958)

Para mejorar la estabilidad y eficiencia del entrenamiento, además de configurar correctamente los tres elementos mencionados anteriormente, se recomienda dividir los datos en batches mediante *DataLoader* con la librería de *Torch*.

El *DataLoader* divide los datos en subconjuntos de datos (batches), esto aporta las siguientes ventajas:

- Reduce el uso de memoria durante el entrenamiento (se deben cargar menos datos a la vez).
- Reduce el sobreajuste, al calcular el gradiente de cada *batch* se guarda la varianza en el cálculo del gradiente.
- Acelera la convergencia, esto pasa porque se realizan actualizaciones más frecuentes.

Para usar *batches* en la red neuronal hace falta modificar el proceso de entrenamiento de la red. En lugar de procesar todos los datos a la vez, el modelo tiene que entrenar *batch* por *batch* dentro de cada época.

Esto también afecta al sistema de evaluación, ahora se tienen que acumular las métricas de cada *batch* y luego agregarse para obtener las métricas de cada época.

## 8.2.2 Recurrent Neural Network (RNN)

El orden de las palabras es muy importante para transmitir el significado de la frase. No es lo mismo “Al niño le mordió el perro” que “Al perro le mordió el niño” o que “Perro el al niño le mordió”, es decir, el orden de las palabras es esencial para entender el significado de lo que se está transmitiendo.

En el año 1986, Michael I. Jordan, publicó un artículo llamado “Serial order: a parallel distributed processing approach”<sup>46</sup>. Allí se propuso añadir conexiones de realimentación (*recurrent links*) para dotar a la red de memoria dinámica y así poder tener en cuenta la secuencia de palabras.

Los resultados que se obtuvieron al aplicar la arquitectura RNN fueron muy buenos, se consiguió tener en cuenta la secuencia de las palabras de manera exitosa. Con los años se propusieron alternativas a la arquitectura RNN tradicional con la intención de mejorar la captación de la secuencia. Una de las arquitecturas alternativas fue la *Gated Recurrent Unit* (GRU)<sup>47</sup>.

La arquitectura de GRU captura las dependencias de las secuencias de forma más compleja gracias a, principalmente, estos dos elementos:

- Cálculo de actualización: decide cuánto del estado anterior de debe conservar.
- Cálculo de reinicio: controla cuánta información pasada se ignora al calcular el nuevo estado.

Estos elementos consiguen extraer los valores de la secuencia y conseguir que el modelo tenga “memoria” de los elementos.

En la siguiente imagen se puede ver la arquitectura de una RNN tradicional y de una GRU:

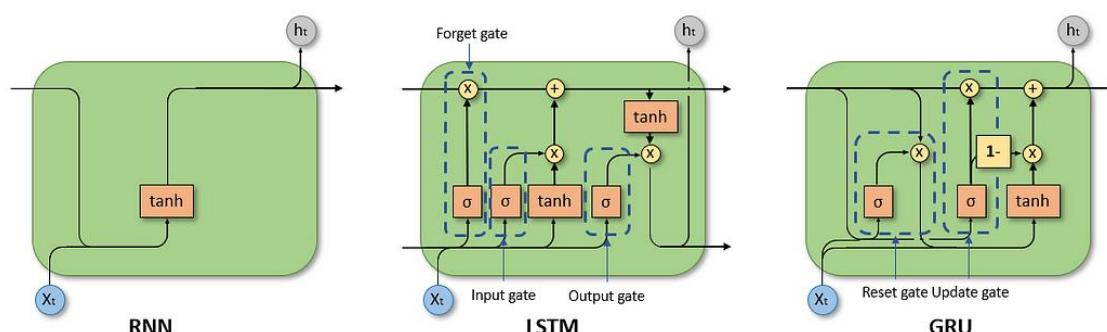


Ilustración 49: arquitectura de RNN y GRU<sup>48</sup>

A parte de las GRU, también hay arquitecturas como *Long Short Term Memory* (LSTM). En este trabajo se han utilizado exclusivamente las GRU, el motivo es que, hay estudios que han demostrado que las GRU consiguen resultados similares a las LSTM, pero con un coste inferior (debido a la menor complejidad de cálculo de las GRU).<sup>49</sup>

<sup>46</sup> (Jordan, 1986)

<sup>47</sup> (Abushnama, 2023)

<sup>48</sup> (Dancker, 2022)

<sup>49</sup> (Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio, 2014)

### 8.2.3 MLP con BERT

Las siguientes arquitecturas se han desarrollado utilizando como datos los *embeddings* que ha generado BERT con los tweets.

#### 8.2.3.1 Arquitectura #1: MLP

La primera arquitectura que se ha creado consta de las capas:

1. 768 nodos (cantidad de *embeddings* de BERT)
2. Capa densa de 512 nodos
3. Capa densa de 256 nodos
4. Capa densa de 128 nodos
5. Capa densa de 32 nodos
6. 1 nodo (0 indicando que no hay odio y 1 indicando que sí).

La función de cálculo de perdida elegida es *BCELoss*, esto es porque, esta función retorna una función sigmoide (es lo que queremos para hacer clasificación binaria) y es muy eficiente, haciendo que sea una buena alternativa para este tipo de problema.

El optimizador que se ha elegido es Adam con un *learning rate* de 0.005

Los resultados de esta primera iteración del *Multi-Layer Perceptron* son los siguientes:

- El valor de F1 es de aproximadamente 0.54
- El progreso del *accuracy* y del error de pérdida del modelo durante el entrenamiento son:

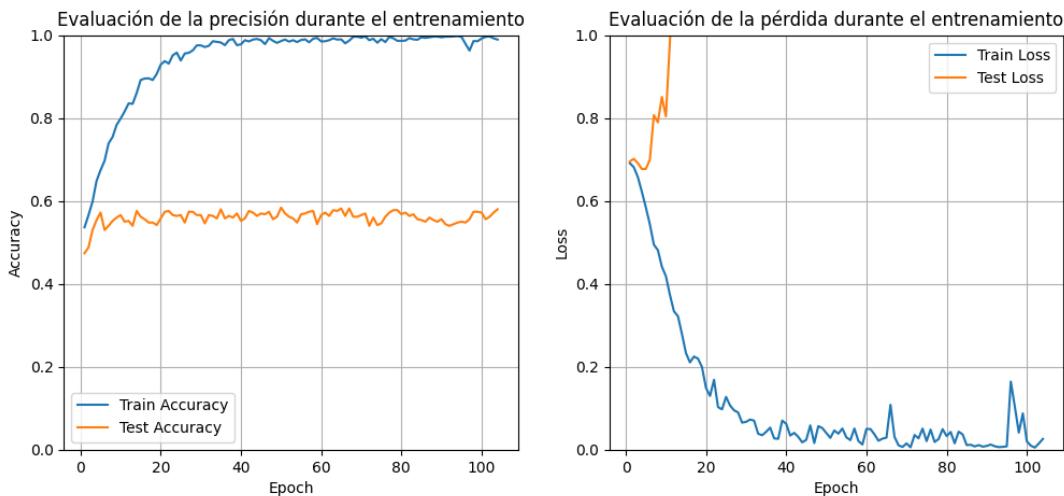


Ilustración 50: Gráfica accuracy y loss MLP

En los resultados de esta gráfica se puede ver como la red se sobreajusta al entrenarse, esto hace que el test se quede estancado en aproximadamente el 0.5. Si miramos la gráfica de los errores podemos ver que el error del entrenamiento tiene una tendencia a la baja mientras que el error de test se dispara rápidamente por encima del 1.

- Si miramos la matriz de confusión se puede ver:

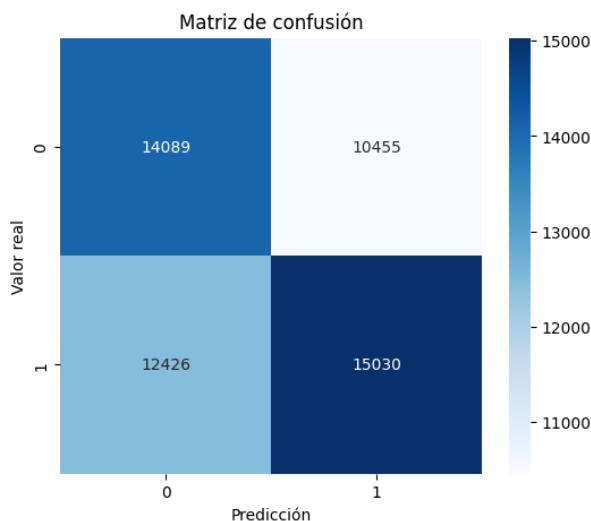


Ilustración 51: Matriz de confusión MLP

Si miramos los resultados de la matriz de confusión podemos ver que los resultados de este modelo no son muy buenos, hay muchos valores que son clasificados erróneamente.

Si se analizan los resultados de este modelo se puede a simple vista ver que hay varias posibles mejoras que podrían incrementar el rendimiento del modelo.

- Al ver las métricas del *accuracy* durante el entrenamiento se puede ver como la fase de entrenamiento esta sobreajustada, esto puede hacer que el modelo “memorice” las respuestas, esto hace que el modelo falle las predicciones en la fase de test ya que las predicciones que se pide que hagan no son las mismas que las que ha memorizado. Una manera de mejorar esta red consistiría en eliminar el sobreajuste, de este modo aprendería a generalizar y así se mejoraría la fase de entrenamiento. Para conseguir esto se utilizará *Dropout*.

*Dropout* es un mecanismo que desactiva aleatoriamente algunas neuronas durante el proceso de entrenamiento, de este modo se fuerza a la red neuronal a aprender con todas las neuronas de la capa y así no sobreajustarse a los datos.

- Para mejorar la clasificación de los datos se cambiará la última capa de la red neuronal por un *Support Vector Machine (SVM)* de este modo se espera conseguir encontrar la frontera entre los datos de mejor manera.

### 8.2.3.2 Arquitectura #2: MLP con dropout

En esta segunda arquitectura se pretende abordar el problema del sobreajuste de la fase de entrenamiento.

La arquitectura que se ha creado consta de las capas:

1. 768 nodos (cantidad de *embeddings* de BERT)
2. *Dropout* del 70%
3. Capa densa de 512 nodos
4. *Dropout* del 50%
5. Capa densa de 256 nodos
6. *Dropout* del 50%
7. Capa densa de 128 nodos
8. Capa densa de 32 nodos
9. 1 nodo (0 indicando que no hay odio y 1 indicando que sí).

Las capas de *Dropout* elegidas tienen un valor muy alto (de 70% y 50%), se han elegido estos valores ya que son los que han conseguido bajar el sobreajuste de la fase de entrenamiento de la manera más consistente.

La función de cálculo de perdida y el optimizador son los mismos que en la arquitectura anterior, *BCELoss* y Adam con un *learning rate* de 0.005

Los resultados conseguidos son los siguientes:

- El valor de F1 es de aproximadamente 0.55
- El progreso del *accuracy* y del error de pérdida del modelo durante el entrenamiento son:

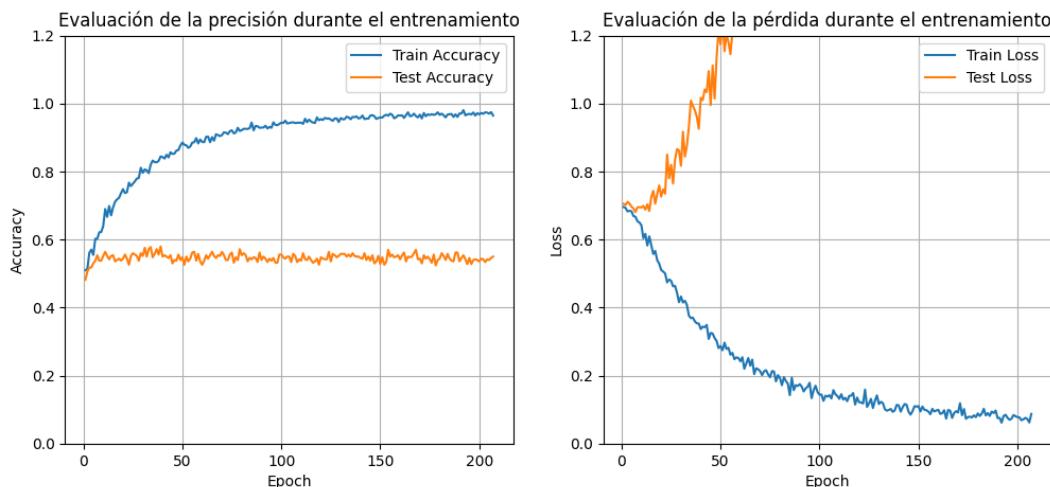


Ilustración 52: Gráfica accuracy y loss MLP con dropout

Se puede ver cómo se ha conseguido bajar el sobreajuste de la fase de entrenamiento de manera exitosa, pero esto no ha ayudado a incrementar los resultados en la capa de test. También se puede ver que el error de evaluación sigue siendo muy elevado y el error de entrenamiento desciende más lentamente.

- Si miramos la matriz de confusión se puede ver:

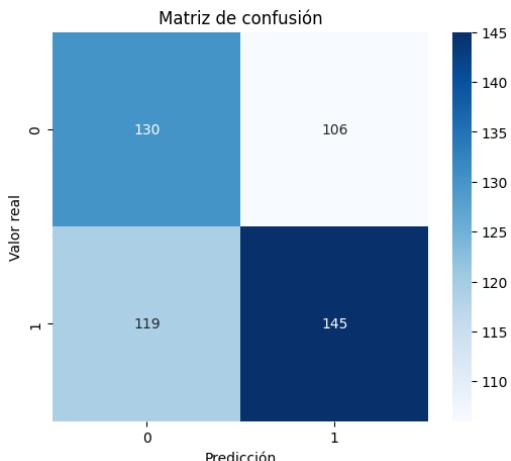


Ilustración 53: Matriz de confusión MLP con dropout

Si miramos los resultados de la matriz de confusión podemos ver que los resultados siguen siendo malos, hay muchos valores que son clasificados erróneamente.

Si analizamos los resultados de las gráficas de accuracy y de pérdida de la red neuronal se puede ver que teniendo un *dropout* tan elevado la red se sigue sobreajustando. Para probar de bajar el sobreajuste sin seguir aumentando el *dropout* se va a reducir la cantidad de neuronas del modelo.

Uno de los motivos por los que se ha llegado a esta conclusión es que al cambiar los valores de *dropout* solo consigue que la curva del *accuracy* de la fase de aprendizaje crezca más lentamente, sigue habiendo sobreajuste. Si miramos los resultados que se consiguen con una red con capas con un *dropout* mucho inferior podemos ver:

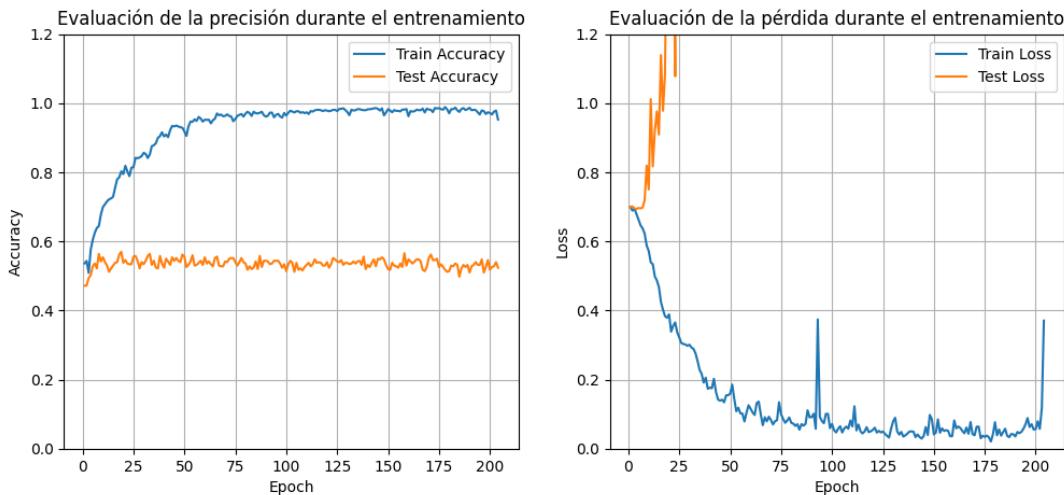


Ilustración 54: MLP con dropout bajo

#### 8.2.3.3 Arquitectura #3: MLP reducido con dropout

En esta tercera arquitectura se pretende profundizar más en el problema del sobreajuste de la fase de entrenamiento.

La arquitectura que se ha creado consta de las capas:

1. 768 nodos (cantidad de *embedings* de BERT)
2. *Dropout* del 60%
3. Capa densa de 256 nodos
4. *Dropout* del 40%
5. Capa densa de 64 nodos
6. Capa densa de 16 nodos
7. 1 nodo (0 indicando que no hay odio y 1 indicando que sí).

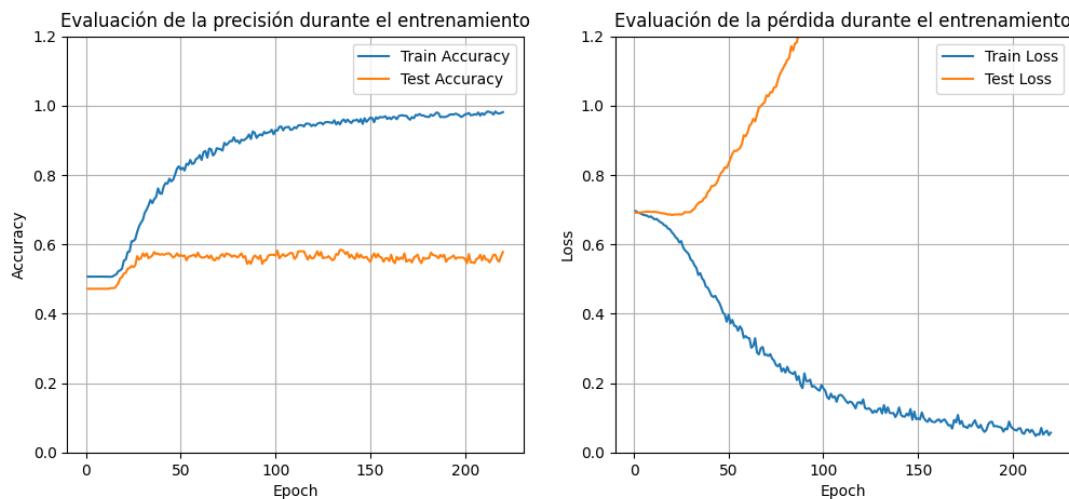
Adicionalmente a estos cambios se ha modificado la función de pérdida, se ha cambiado de *BCELoss* a *BCEWithLogitsLoss*. Se ha hecho este cambio con la intención de mejorar las predicciones con una función más compleja que la anterior. En cuanto al optimizador, sigue siendo Adam, pero con un *learning rate* de 0.00005.

Para mejorar el entrenamiento se ha modificado el tamaño de los *batches* del *DataLoader* de 64 a 16.

Estos cambios han conseguido mejorar el modelo de los siguientes puntos:

- El valor de F1 ha aumentado de 0.55 a 0.59.

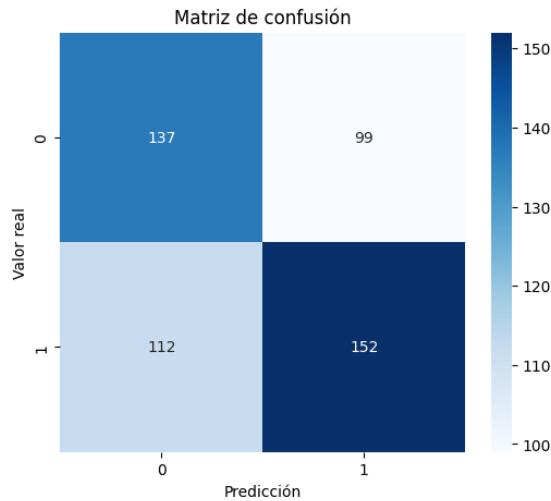
- El progreso del **accuracy** y del error de pérdida del modelo durante el entrenamiento son:



*Ilustración 55: Gráfica accuracy y loss MLP reducido con dropout*

Se puede ver cómo se ha conseguido bajar el sobreajuste de la fase de entrenamiento de manera exitosa y aumentar el porcentaje de entrenamiento de los datos en la fase de test. También se puede ver que se ha suavizado el error durante el entrenamiento

- Si miramos la matriz de confusión se puede ver:



*Ilustración 56: Matriz de confusión MLP reducido con dropout*

Se puede ver que han aumentado la cantidad de valores bien clasificados y han disminuido la cantidad de valores mal clasificados respecto a los modelos anteriores.

#### 8.2.3.4 Arquitectura #4: MLP #3 + SVM

Esta arquitectura intenta mejorar los resultados de la clasificación combinando la red neuronal 3 (“Arquitectura #3: MLP reducido con dropout”) con una SVM.

Las métricas conseguidas son las siguientes:

- El valor de F1 ha aumentado de 0.59 a 0.61.
- Si miramos la matriz de confusión se puede ver:

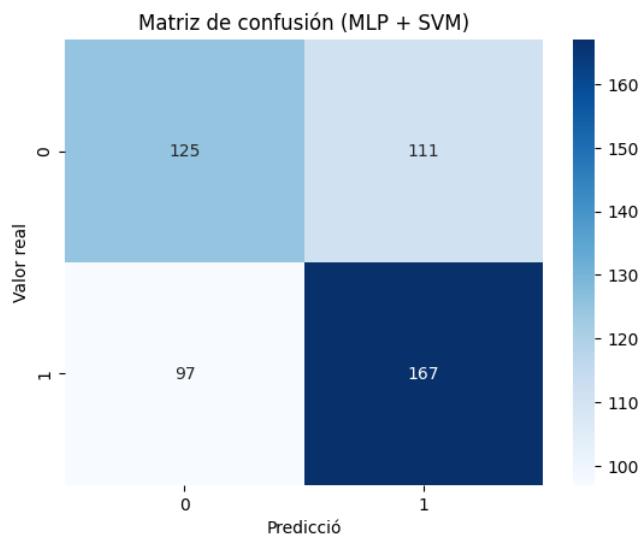


Ilustración 57: Matriz de confusión MLP arquitectura #3 + SVM

Si se comparan los resultados de este modelo se puede ver que han aumentado la cantidad de elementos con odio clasificados como odio. El problema de estos resultados es que no se han aumentado porque el modelo haya encontrado una frontera mejor entre odio y no odio, han aumentado porque en general clasifica los tweets como odio en general.

Viendo que usar SVM ha incrementado un poco los resultados se han aplicado *GridSearch* para optimizar los parámetros. Los parámetros que se han probado son:

- “C”: Este parámetro indica el error permitido durante el entrenamiento. Al probar con valores grandes y pequeños se encuentra un valor balanceado entre el *bias* y varianza.
  - Valores probados: 0.01, 0.1, 1, 10 i 100
- “Kernel”: Indica el método de separación de espacios.
  - Valores probados: “lineal”, “rbf” i “polly”

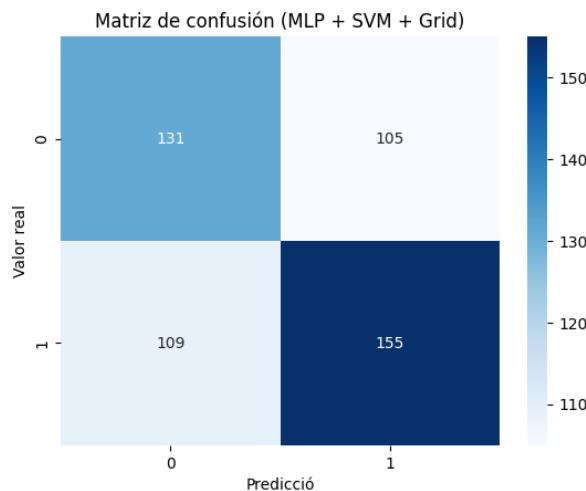
- “Gamma”: Define como de influyente se debe considerar un punto que se encuentre solo.
  - Valores probados: “scale” i “auto”
- “Degree”: Es el grado polinómico que tiene el kernel para encontrar una línea que separe los grupos.
  - Valores probados: 2, 3 i 4

Los resultados de la optimización fueron:

- “C”: 0.01
- “Kernel”: linear
- “Gamma”: scale
- “Degree”: 2

Los resultados conseguidos con esta configuración son los siguientes:

- El valor de F1 ha disminuido de 0.61 a 0.59
- Si miramos la matriz de confusión se puede ver:



*Ilustración 58: Matriz de confusión MLP arquitectura #3 + SVM + GridSearch*

Aunque el modelo tenga un valor F1 inferior al anterior, se ha conseguido que el modelo falle “hacia el otro lado”, es decir que se priorice más al no odio en las predicciones.

### 8.2.4 MLP con BETO

Para ver si el problema de los malos resultados de clasificación en los textos de odio era de los datos o de los modelos se pensó probar dicho *dataset yelp\_polarity* en las arquitecturas descritas anteriormente, el *dataset* contiene *reviews* de productos en Yelp, estos comentarios están clasificados en función de si el sentimiento que transmiten es positivo o negativo.

Al ser dos contextos y *datasets* completamente distintos no tiene mucho sentido que realmente se comparan las métricas entre los modelos de manera directa, la intención de esta prueba es la de determinar si la arquitectura es capaz de discernir entre tipos de textos o no.

Los resultados con el nuevo dataset fueron muy buenos, consiguiendo un valor de f1 de aproximadamente 0.9. Viendo que la diferencia de f1 entre los *datasets* es tan distinta se tomó la decisión de probar con un encoder BETO (la versión de BERT para textos en español).

Para hacer que la comparativa entre arquitecturas usando BERT y BETO fueran lo más justas posibles solo se cambió el *encoder*, la cantidad y proporción de datos se conservó. Los resultados de cada modelo se pueden ver en la siguiente tabla:

Modelo	F1	Matriz de confusión																							
MLP	0.57	<p style="text-align: center;"><b>Matriz de confusión</b></p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td colspan="2"></td> <th colspan="2">Valor real</th> </tr> <tr> <td colspan="2"></td> <th>0</th> <th>1</th> </tr> <tr> <th rowspan="2">Valor predicción</th> <th>0</th> <td>157</td> <td>96</td> </tr> <tr> <th>1</th> <td>104</td> <td>143</td> </tr> <tr> <td colspan="2"></td> <td>0</td> <td>1</td> </tr> <tr> <td colspan="2"></td> <td style="text-align: right;">Predicción</td> <td></td> </tr> </table>			Valor real				0	1	Valor predicción	0	157	96	1	104	143			0	1			Predicción	
		Valor real																							
		0	1																						
Valor predicción	0	157	96																						
	1	104	143																						
		0	1																						
		Predicción																							
MLP con <i>Dropout</i>	0.59	<p style="text-align: center;"><b>Matriz de confusión</b></p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td colspan="2"></td> <th colspan="2">Valor real</th> </tr> <tr> <td colspan="2"></td> <th>0</th> <th>1</th> </tr> <tr> <th rowspan="2">Valor predicción</th> <th>0</th> <td>149</td> <td>104</td> </tr> <tr> <th>1</th> <td>93</td> <td>154</td> </tr> <tr> <td colspan="2"></td> <td>0</td> <td>1</td> </tr> <tr> <td colspan="2"></td> <td style="text-align: right;">Predicción</td> <td></td> </tr> </table>			Valor real				0	1	Valor predicción	0	149	104	1	93	154			0	1			Predicción	
		Valor real																							
		0	1																						
Valor predicción	0	149	104																						
	1	93	154																						
		0	1																						
		Predicción																							

Modelo	F1	Matriz de confusión				
MLP reducido con Dropout	0.56	<p style="text-align: center;"><b>Matriz de confusión</b></p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>151</td><td>102</td> </tr> <tr> <td>110</td><td>137</td> </tr> </table>	151	102	110	137
151	102					
110	137					
MLP #3 con SVM	0.57	<p style="text-align: center;"><b>Matriz de confusión (MLP + SVM)</b></p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>146</td><td>107</td> </tr> <tr> <td>104</td><td>143</td> </tr> </table>	146	107	104	143
146	107					
104	143					
MLP #3 con SVM y GridSearch	0.57	<p style="text-align: center;"><b>Matriz de confusión (MLP + SVM + Grid)</b></p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>143</td><td>110</td> </tr> <tr> <td>103</td><td>144</td> </tr> </table>	143	110	103	144
143	110					
103	144					

Si miramos los resultados de las matrices de confusión podemos ver que los resultados de este modelo no son muy buenos, hay muchos valores que son clasificados erróneamente. Si se compara con la misma arquitectura con BERT, se puede ver el modelo conseguido peores resultados.

Viendo que los Multi-layer Perceptrons (MLP) no han conseguido buenos resultados al usar embeddings de BERT se planeó cambiar de arquitectura a una que aproveche el contexto y las secuencias de las palabras. Para hacer esto se decidió cambiar la arquitectura de MLP a una red neuronal recurrente (RNN).

## 8.2.5 RNN con BETO

La arquitectura usada de la RNN es la siguiente:

1. BETO
2. 1 capa GRU de 768 nodos
3. Capa densa de 64
4. *Dropout* de 0.5
5. Capa densa de 2 nodos

El entrenamiento de esta red se hace con 100 epochs, usando un *early stopping* del 10% del total de los epochs (es decir de 10 épocas).

Con esta arquitectura lo que se pretende es ver el efecto de usar BETO entrenado juntamente con una RNN para ver si los resultados mejoran al tener en cuenta la secuencia de las palabras.

Este modelo se ha desarrollado para el artículo adjuntado en el capítulo “Artículo CCIA edición 27ª”. En el artículo se puede ver otras variaciones del modelo.

La propuesta de arquitectura anterior es la que ha ofrecido mejores resultados teniendo la complejidad y el coste más bajos. En modelos con más capas GRU y con capas densas más grandes, se ha visto que la red conseguía resultados similares.

Al probar la arquitectura anterior propuesta con distintos datasets se consiguieron los siguientes resultados:

*Tabla 11: resultados del modelo rnn en los distintos datasets*

Dataset	F1 score	Accuracy	Precision	Recall	ROC-AUC
Youtube (emojis tokenizados)	0.6179 ± 0.1736	0.6179 ± 0.1732	0.6186 ± 0.1717	0.6115 ± 0.3824	0.6179 ± 0.1733
Youtube (sin emojis)	0.6267 ± 0.1195	0.6271 ± 0.1176	0.6214 ± 0.1274	0.6581 ± 0.1678	0.6270 ± 0.1179
Twitter (emojis tokenizados)	0.7157 ± 0.0538	0.7166 ± 0.0521	0.7150 ± 0.2034	0.7451 ± 0.8740	0.7159 ± 0.0535
Twitter (sin emojis)	<b>0.7556 ± 0.1161</b>	<b>0.7558 ± 0.1153</b>	<b>0.7714 ± 0.2059</b>	<b>0.7509 ± 0.4363</b>	<b>0.7559 ± 0.1157</b>
Todo (emojis tokenizados)	0.7060 ± 0.1973	0.7062 ± 0.1958	0.7113 ± 0.4121	0.7099 ± 0.4204	0.7062 ± 0.2002
Todo (sin emojis)	0.7002 ± 0.0786	0.7003 ± 0.0727	0.7158 ± 0.3038	0.6832 ± 0.1641	0.7006 ± 0.0829

*Nota: las varianzas están en escala 10<sup>-3</sup>*

Se puede ver que el mejor resultado es el de Twitter sin emojis. La matriz de confusión que consigue este modelo con el dataset de Twitter sin los emojis es la siguiente:

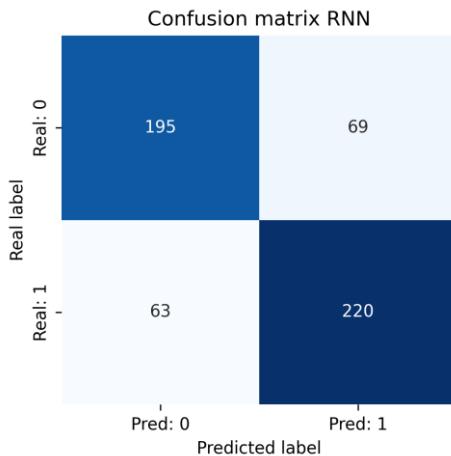


Ilustración 59: matriz de confusión RNN

Si miramos el proceso de aprendizaje de la RNN podemos ver lo siguiente:

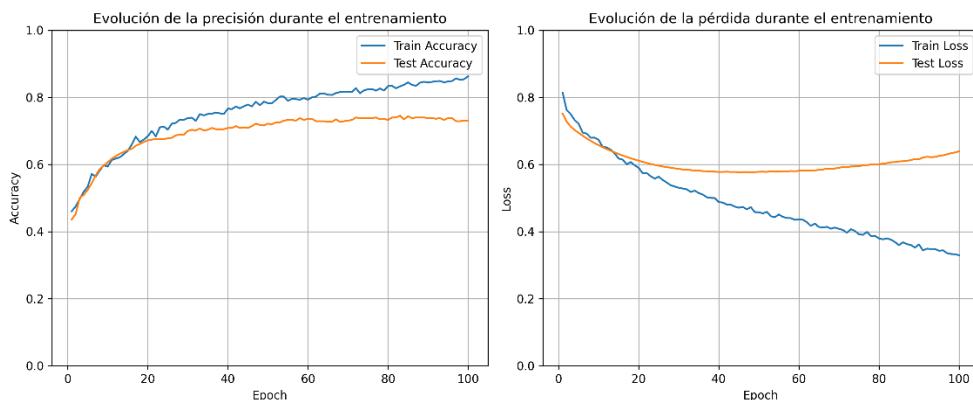


Ilustración 60: proceso de entrenamiento de la RNN

Viendo estos resultados se puede ver que el modelo aprende correctamente. La métrica  $f_1$  es de 0.7556, confirmando que la secuencia de la frase aporta información muy útil al intentar clasificar una frase. Si se analizan las otras métricas conseguidas en el dataset de Twitter se puede ver que indican que el modelo ha aprendido correctamente, no favorece ninguna clase, esto se puede ver con los valores de *recall* y precisión que son parecidos, el valor ROC-AUC indica que el modelo puede distinguir correctamente entre las clases.

Si se analizan las gráficas de entrenamiento se puede ver que el aprendizaje en la fase de entrenamiento es bueno, no hay sobreentrenamiento. En la fase de test se puede ver que el modelo se queda estancado alrededor de 0.75 de *accuracy*, se puede ver que el entrenamiento se ha parado en el epoch 60, esto es a que el *early stopping* se ha activado ya que no se ha detectado ninguna mejora del modelo des de hace 10 epochs.

La matriz de confusión muestra buenos resultados, el modelo no predice clase más que la otra.

Investigando si hay algún patrón en los tweets mal clasificados. A continuación, se muestran algunos ejemplos:

- "Buenos días solo a @MonicaCarrillo □ <https://t.co/thuTTQE7h>"
  - Clasificado como no odio, pero etiquetado como sí. El usuario no está dirigiendo toxicidad hacia Monica Carrillo (aunque sea inapropiado el comentario, esto no es lo que se está investigando).
- "@jesusmarana @MartaMonforteJ No es no,¿ se te ha olvidado?."
  - Clasificado como odio, pero etiquetado como sí. El usuario está haciendo burla.
- "@manuelrico @RosaVillacastin Eres un ser de mar te veo y me repugnas."
  - Está etiquetado como no odio, el clasificador dice que sí que lo es.

Al ver estos tweets se puede llegar a pensar que un posible problema es que el modelo no puede aprender más ya que los datos están etiquetados por varios etiquetadores, eso ha hecho que no se hayan etiquetado los tweets con los mismos criterios exactos.

## 8.2.6 RNN + MLP

Viendo que los tweets se pueden separar en dos elementos, texto y emojis, se decidió crear una estructura la cual tuviera en cuenta simultáneamente tanto la secuencia de palabras como los emojis.

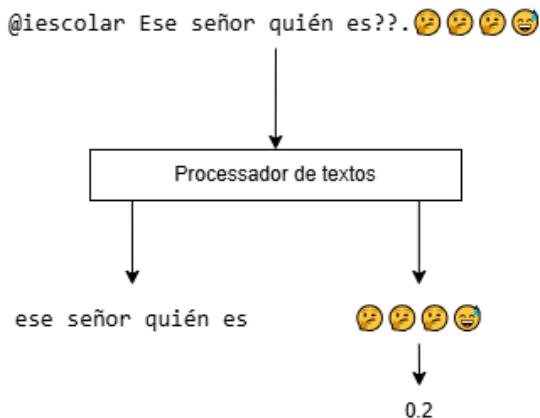
Para poder procesar la parte de los textos se decidió usar una Red Neuronal Recurrente (RNN) ya que, gracias a su estructura, tiene en cuenta la secuencia de las palabras mejor que otros tipos de redes neuronales. El problema de utilizar una RNN directamente es que los emojis añaden mucho ruido a la estructura del texto. Para ilustrar el problema que se plantea se puede **ver** el siguiente ejemplo:

*Tabla 12: ejemplo de tokenización de emojis*

<b>Tweet original</b>	Con Ortega Smith de V <small>呕</small> X no pediste lo mismo
<b>Tweet procesado</b>	con ortega smith de V:face_vomiting:X no pediste lo mismo

Como se puede ver, tanto en el texto original como en el procesado, el emoji no aporta ninguna información sintáctica que pueda ayudar en la búsqueda de patrones en las palabras. Al ser un ícono no aporta realmente el mismo tipo de información que una palabra, una palabra aporta información sintáctica mientras que un emoji aporta una información más enfocada a enfatizar o a resaltar la emoción que transmite el mensaje.

Para poder hacer que el modelo enfatice más en la relación entre los emoticonos y los textos se decidió hacer el siguiente pipeline de procesado de datos:



*Ilustración 61: pipeline de procesado de datos*

Como se puede ver en la ilustración anterior, los emojis pueden repetirse. Sabiendo que esto es porque el emisor del texto quiere enfatizar la emoción, eliminar los duplicados dejando solamente un emoji no es una opción viable.

Para calcular la puntuación de los emojis en un texto se ha usado la siguiente fórmula:

Ecuación 7: fórmula de la puntuación de los emojis

$$\text{Puntuación emoji} = \sum_{n=0}^n f(n) * s(n)$$

Donde  $f(n)$  es la frecuencia del emoji en el texto y  $s(n)$  es la puntuación (score) del emoji.

En el ejemplo anterior vemos que el tweet tiene los siguientes emojis: 😢 😢 😢 😊

Al aplicar la fórmula nos queda:  $0.0 * 3 + 0.2 * 1$

Si miramos otros ejemplos:

Tabla 13: ejemplos de la aplicación de la ecuación 1

<b>Tweet</b>	@lago_jorge Mucho orgullo ❤️ ❤️ ❤️
<b>Emojis</b>	❤️ ❤️ ❤️
<b>Fórmula</b>	$0.6 * 3 = 1.8$

<b>Tweet</b>	@iescolar Noooo !!!Ada Kolau no podrà anar a Gaza perquè no tindrà visat ???👤👤👤👤⚡⚡⚡⚡
<b>Emojis</b>	👤👤👤👤⚡⚡⚡⚡
<b>Fórmula</b>	$0.0 * 3 + 0.8 * 3 = 2.4$

<b>Tweet</b>	@iescolar Felipe González es un mierda 💩 todo los sabemos 🤢🤮🤮🤮🤮
<b>Emojis</b>	💩🤮🤮🤮🤮🤮
<b>Fórmula</b>	$-0.2 * 1 + -0.6 * 5 = -3.2$

Nota: El segundo ejemplo es en catalán, en el preprocessado se eliminan las lenguas que no sean castellano ya que no se está creando un modelo multilengua. Se ha añadido ya que es un buen ejemplo.

Una vez se ha extraído el texto y se ha asociado una puntuación de emojis al tweet se va a hacer el siguiente modelo:

- Primero introducimos a BETO el texto del tweet:

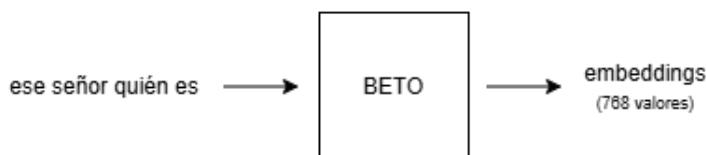


Ilustración 62: diagrama procesado BETO

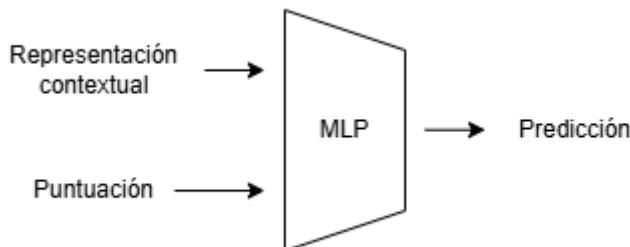
- A continuación, entrenamos una red neuronal recurrente con los embeddings conseguidos con BETO.



*Ilustración 63: diagrama entrenamiento RNN con embeddings BETO*

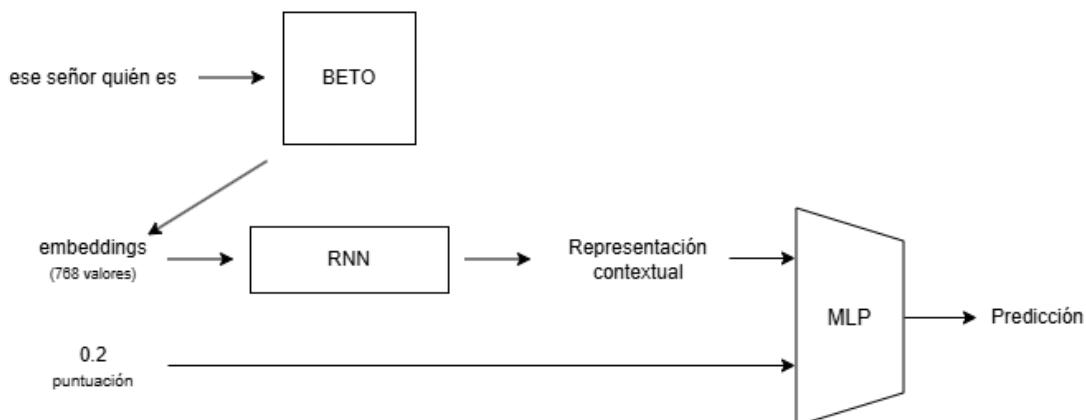
Una vez conseguida la representación contextual del mensaje se combina juntamente con el score en un Multi-Layer Perceptron (MLP).

- Se entrena el MLP con todos los datos.



*Ilustración 64: diagrama estructura MLP con representación contextual y puntuación*

El proceso entero es el siguiente:



*Ilustración 65: diagrama proceso entrenamiento RNN+MLP*

Una vez diseñado el procesamiento de los datos y la arquitectura del modelo, se tuvo que desarrollar el código.

El modelo se ha entrenado con 50 épocas, teniendo un *early stopping* del 10%.

El modelo de BERT usado fue el de la versión en español. Se han entrenado solamente con 1 de las 12 capas.

La estructura de la Red Neuronal Recurrente (RNN) es la siguiente:

1. 1 capa GRU de 768 nodos
2. Capa densa de 64
3. Dropout de 0.5
4. Capa densa de 2 nodos

El valor f1 conseguido con la arquitectura anterior en el dataset de twitteres de: 0.7623

La estructura de la Multi-Layer Perceptron (MLP) continuaba a partir de la salida de la última capa de la RNN, es decir la entrada de la primera capa es de 64 nodos (del output de la capa densa del RNN) + 1 nodo (puntuación emojis).

La arquitectura del MLP es la siguiente:

- Capa densa de 32 nodos
- Dropout de 0.4
- Capa densa de 2 nodos

Los resultados del entrenamiento incrementaron en comparación con la arquitectura de la RNN con embeddings de BETO. Si miramos la siguiente tabla podemos ver los resultados del modelo al clasificar los distintos datasets:

*Tabla 14: resultados del modelo rnn+mlp en los distintos datasets*

<b>Dataset</b>	<b>F1 score</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>ROC-AUC</b>
Youtube (emojis tokenizados)	0.6516 ± 0.8303	0.6550 ± 0.6091	0.6607 ± 1.050	0.6619 ± 10.5930	0.6549 ± 0.6082
	0.6696 ± 0.3628	0.6709 ± 0.3838	0.6652 ± 0.5280	0.7041 ± 3.219	0.6706 ± 0.3814
Twitter (emojis tokenizados)	0.7629 ± 0.3996	0.7645 ± 0.3878	0.7705 ± 1.0615	0.7839 ± 5.0560	0.7638 ± 0,3708
	0.7912 ± 0.2643	0.7933 ± 0.2449	0.7850 ± 2.1803	0.8334 ± 5.1001	0.79365 ± 0.3679

Todo (emojis tokenizados)	$0.7714 \pm 0.0529$	$0.7721 \pm 0.0519$	$0.7724 \pm 0.3920$	$0.7929 \pm 0.9911$	$0.7715 \pm 0.0553$
Todo (sin emojis)	$0.7728 \pm 0.2211$	$0.7737 \pm 0.2353$	$0.7738 \pm 0.2200$	$0.7958 \pm 3.3013$	$0.7730 \pm 0.2008$

Nota: las varianzas están en escala  $10^{-3}$

La mejor f1 conseguida es de  $0.7912 \pm 0.2643 * 10^{-3}$  con el dataset de Twitter. Si miramos este caso, se puede ver que todos los datos están situados alrededor de 0.79, el valor del *recall* es superior, llegando hasta el 0.8334, al compararlo con el valor de precisión y de f1 se puede concluir que el modelo no tiene preferencia por ninguna clase.

Viendo estos resultados se puede concluir que los emojis tienen una relevancia alta dentro del proceso de clasificación de odio en tweets ya que este modelo mejora respecto a los que no procesan los emojis. También se puede ver que el modelo clasifica peor cuando los emojis son tokenizados en vez de solamente procesarse por separado.

Si miramos el proceso de entrenamiento:

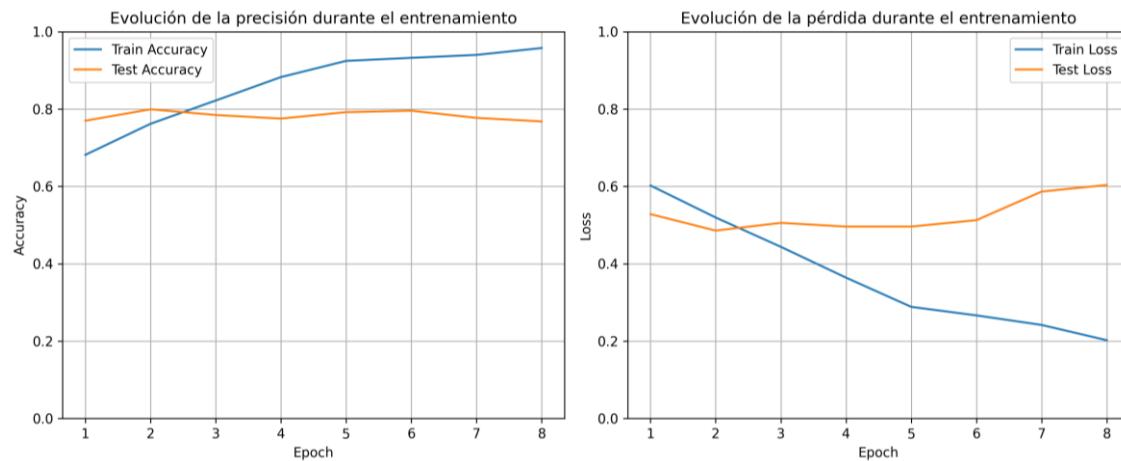


Ilustración 66: proceso de entrenamiento de la RNN+MLP

Si miramos la matriz de confusión:

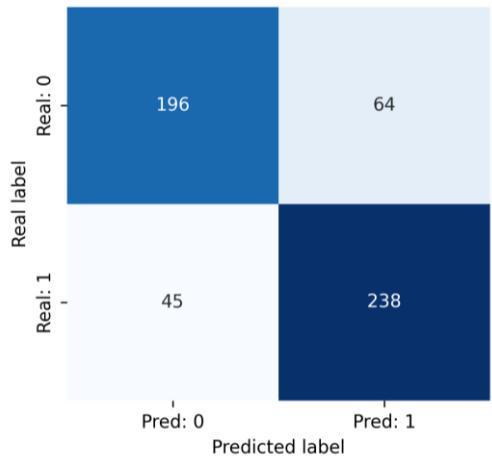


Ilustración 67: matriz de confusión RNN+MLP

Este modelo clasifica mejor que la RNN, se puede concluir que los emojis ayudan a la clasificación de los tweets de manera exitosa.

Analizando los resultados mal clasificados se han encontrado tweets como los siguientes:

- "@CarmeBarcelo Que comentario tan inútil y que denota un vacío interior propio de una amargada del fútbol.. y me quedo corto 😞"
  - Clasificado como no odio, pero la etiqueta dice que sí que contiene odio.
- "@MartaMonforteJ 🔴Pero como es posible que la respuesta del tal Honrubia tenga 5 veces mas likes que el tuit original de la COLABORADORA DE INFRALIBRE! 😂🤦‍♀️😂"
  - El modelo ha predicho que este tweet no tiene odio, pero la etiqueta indica que si

Al ver los resultados se puede ver que la cantidad de tweets con emojis mal clasificados se reduce bastante, el problema está cuando los emojis y el texto transmiten información diferente, en estos casos el modelo no sabe qué hacer.

### 8.2.7 RNN + MLP por separado

Una vez visto que los modelos mejoran al procesar conjuntamente el texto y los emojis, se investigó cuánto aporta cada elemento al clasificador. Es decir, se decidió separar el mejor modelo que se había hecho hasta el momento (el del apartado “RNN + MLP”) en la parte de procesamiento de texto y la parte de procesamiento de emojis.

Para hacer esto se crearon dos modelos, BETO + MLP y un modelo de clasificación por puntuación de emojis. Se utilizó el dataset de comentarios de Twitter ya que fue el que consiguió mejores resultados en el modelo RNN + MLP.

El modelo de BETO+MLP es similar al del apartado anteriormente (“MLP con BETO”) solamente que esta vez se ha usado una arquitectura de MLP más compleja y se han entrenado 6 de las 12 capas de BETO. El diagrama de la arquitectura genérica del modelo propuesto es el siguiente:

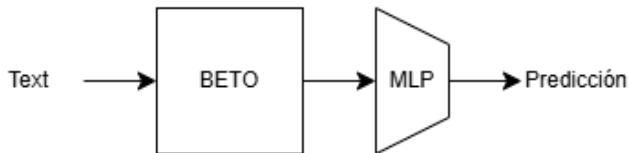


Ilustración 68: BETO con MLP

La arquitectura del MLP es la siguiente:

- Capa densa de 128 nodos
- Dropout de 0.4
- Capa densa de 32 nodos
- Dropout de 0.2
- Capa densa de 2 nodos

La f1 conseguida en este modelo es de 0.7730, un accuracy de 0.7773 y una loss de 0.9737.

Se puede ver que el modelo ha conseguido mejores resultados que la RNN, esto se debe a que el modelo de la RNN recibe embeddings de BETO mientras que la arquitectura de BETO + MLP entrena a BETO, consiguiendo embeddings de manera dinámica.

Si se mira la matriz de confusión se puede ver que los resultados son muy estables, no hay ninguna categoría que tenga muchas más predicciones que la otra:

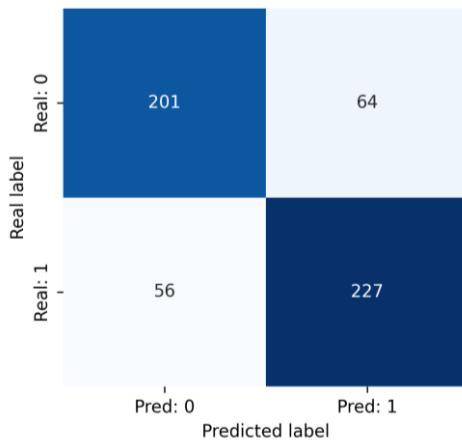


Ilustración 69: matriz de confusión de BETO+MLP

Viendo estos resultados se puede llegar a la conclusión que, por lo menos, los textos tienen una importancia muy alta.

Para evaluar la importancia de los emojis se han hecho tipos de experimento. El primero, es el más sencillo y consistió en asignar la puntuación en función de la puntuación de los emojis directamente.

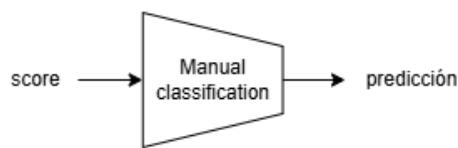
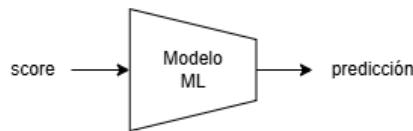


Ilustración 70: clasificación manual en función de la puntuación de los emojis

Los resultados conseguidos fueron malos, la f1 conseguida fue de 0.4924, descartando directamente este método como bueno para la clasificación de los tweets.

El segundo experimento consistió en introducir la puntuación de los emojis a un modelo para hacer la predicción



*Ilustración 71: clasificación con ML en función de la puntuación de los emojis*

Para poder hacer la comparativa se han utilizado los tres mismos modelos que en el apartado de Machine learning (ML), es decir, SVM, RandomForest y XGBoost, pero este caso utilizando la puntuación de los emojis como variable para entrenar. Los resultados son los siguientes:

*Tabla 15: clasificación de los modelos en función de la puntuación de los emojis*

Modelo	F1 score	Accuracy	Roc Auc curve
SVM	0.357921	0.518709	0.498463
RandomForest	0.369250	0.494679	0.513499
XGBoost	0.354715	0.519053	0.500000

Se puede ver que la clasificación basada solamente en la puntuación de los emojis no es un buen método de clasificación. Esto se debe a dos motivos principales:

1. Muchos comentarios no tienen emoticonos, haciendo que muchos de los datos de clasificación tengan una puntuación de 0. Esto hace que la clasificación se haga prácticamente de manera aleatoria.
2. Los emojis se pueden utilizar irónicamente o de modo ambiguo, pueden no reflejar la intención del mensaje. Por ejemplo: puede haber un comentario clasificado como odio ya que contiene burla, pero los emojis son positivos ya que el emisor los ha usado para reírse (con 😂 o 😅).

Estos dos puntos indican que la clasificación de los textos con solamente los emojis no sea buena debido a la falta de información del contexto del mensaje. Aun así, pueden usarse como método complementario para dar más contexto en la clasificación.

### 8.2.8 RNN + MLP con emojis codificados con one-hot encoding

Este modelo se creó con la intención de mejorar el mejor modelo que se ha conseguido hasta el momento, el RNN + MLP. La diferencia clave reside en el método de procesado de los emojis. Previamente se utilizaba la fórmula:

$$\text{Puntuación emoji} = \sum_{n=0}^n f(n) * s(n)$$

Viendo esto se pensó que puede ser que con solamente un número el modelo no puede realmente valorar la intencionalidad con los emojis.

Para aumentar la cantidad de información que se le da al modelo sobre los emojis se decidió crear utilizar la técnica de one-hot encoding (explicada en el capítulo “One-hot encoding”).

Al añadir la técnica de one-hot encoding para procesar los emojis se consigue la siguiente estructura:

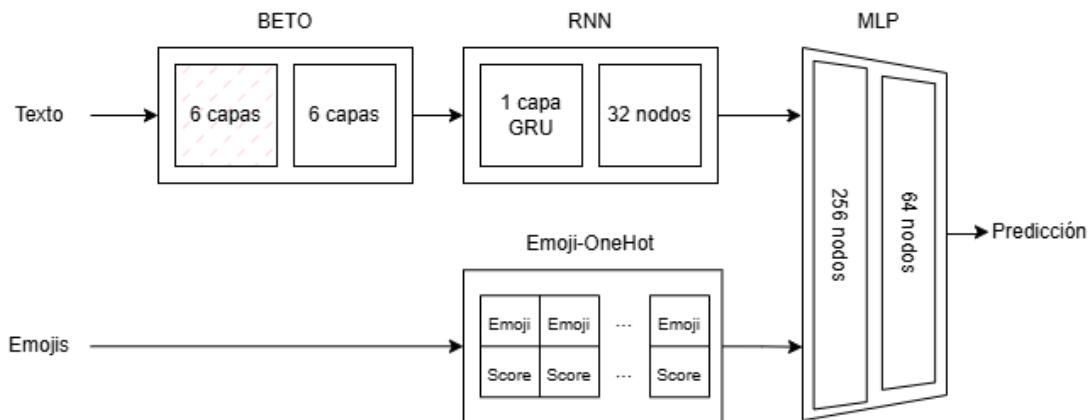


Ilustración 72: estructura RNN+MLP+OneHotEmoji

El texto se introduce a BETO el cual entrena con 6 de sus 12 capas, el output de BETO se introduce a una RNN la cual tiene 1 capa de GRU y 32 nodos de capa oculta. Finalmente se introduce el output de la RNN y el vector de los emojis con one-hot encoding a un MLP que tiene 2 capas densas internas, la primera reduce la dimensionalidad de 250 a 128 nodos y la segunda capa oculta reduce de 128 a 32, finalmente se consigue una predicción sobre si el tweet contiene toxicidad o si no.

Este modelo consiguió los siguientes resultados en los dataset probados:

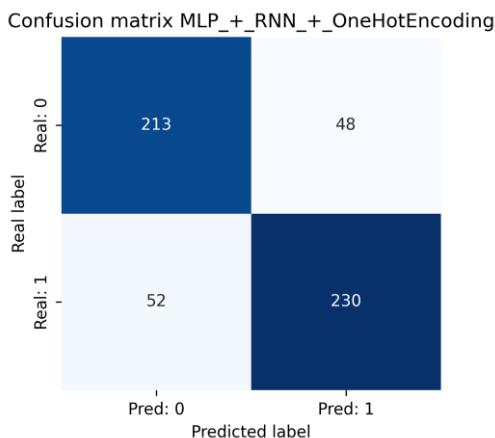
*Tabla 16: resultados del modelo rnn+mlp+one-hot en los distintos datasets*

Dataset	F1 score	Accuracy	Precision	Recall	ROC-AUC
Youtube (emojis tokenizados)	0.6469 ± 0.1865	0.6496 ± 0.2025	0.6599 ± 0.6970	0.6393 ± 9.6150	0.6497 ± 0.1939
Youtube (sin emojis)	0.6711 ± 1666	0.6474 ± 0.1876	0.6649 ± 0.1170	0.7092 ± 3.6758	0.6721 ± 0.1786
Twitter (emojis tokenizados)	<b>0.7959 ± 0.1828</b>	<b>0.7973 ± 0.1537</b>	<b>0.7861 ± 1.1546</b>	<b>0.8382 ± 1.4157</b>	<b>0.7959 ± 0.1831</b>
Twitter (sin emojis)	0.7564 ± 0.2204	0.7571 ± 0.2023	0.7654 ± 0.7728	0.7671 ± 1.4164	0.7568 ± 0.2190
Todo (emojis tokenizados)	0.7714 ± 0.0529	0.7721 ± 0.0519	0.7724 ± 0.3929	0.7929 ± 0.9911	0.7715 ± 0.0553
Todo (sin emojis)	0.7728 ± 0.2211	0.7737 ± 0.2353	0.7738 ± 0.2200	0.7958 ± 3.301	0.7729 ± 0.2008

Nota: las varianzas están en escala  $10^{-3}$

Se puede ver que los mejores resultados conseguidos por este modelo son los que usan el dataset de Twitter con los emojis tokenizados como parte del texto, con un f1 score de  $0.7959 \pm 0.1828$ . Al igual que el modelo anterior, el valor más alto es la recall (0.8382), pero se puede que la f1 (0.7959) y la precisión (0.7861) indiquen que el modelo tiene un buen equilibrio entre las clases, no muestra ninguna preferencia clara por ninguna clase. Además, el alto valor del ROC-AUC confirma que el modelo distingue adecuadamente entre las clases.

La matriz de confusión del mejor resultado es la siguiente:



*Ilustración 73: matriz de confusión del modelo de MLP+RNN con one-hot encoding*

Si se mira el proceso de entrenamiento se puede ver:

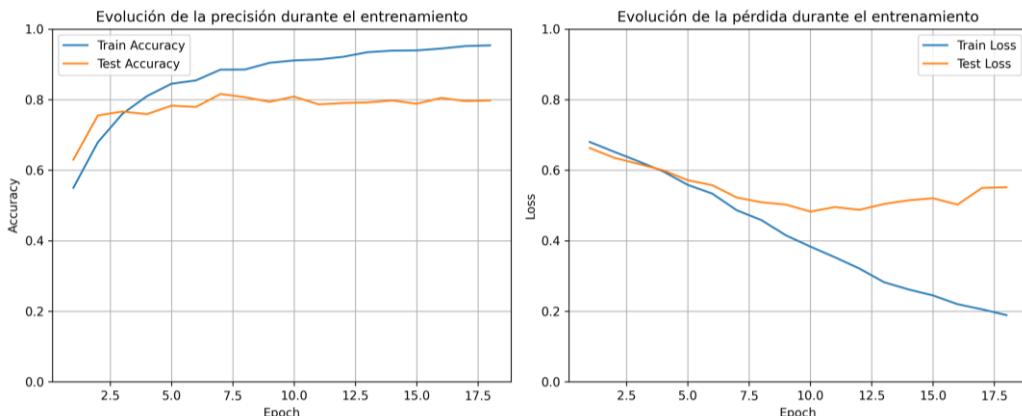


Ilustración 74: entrenamiento de RNN+MLP con emojis con one-hot encoding

Como se puede ver, el modelo aprende en las primeras épocas, pero luego se queda estancado. El crecimiento crece rápidamente hasta llegar al séptimo epoch, luego el modelo deja de aprender y es detenido por el *early stopping* en el epoch 18. En cuanto al error, se advierte cómo empieza a crecer en el momento en el que el modelo deja de aprender, y al superar el epoch 7 el modelo empieza a acumular errores en la fase de test.

Si se analizan los valores mal clasificados se puede ver comentarios como los siguientes:

- "Esto se debe al multiculturalismo 😂😂😂😂 que dejamos entrar a todo el 🌎 sin condiciones sin normas y todo because and Happy, me your feel😡😡😡 xd"
  - a. Clasificado como no odio, etiquetado como odio
- "Estos hijos de su madre me dan risa xq no creo que le haga gracia que se le metan a su casa 👤👤👤"
- "Me encanta como esta subespecie demuestra claramente como funciona la selección natural..."
- a. Clasificado como no odio y etiquetado como odio.

Los resultados indican que la representación más compleja de los emojis ha mejorado la clasificación de textos con emojis, la cantidad de comentarios mal clasificados con emojis ha bajado. En el primer ejemplo, se puede ver como los emojis contradicen al sentimiento del texto, esto ha confundido al modelo. Si se miran más ejemplos se puede confirmar que el modelo no entiende correctamente el comentario cuando el sentimiento del texto y del emoji indican emociones distintas.

Se puede ver que el modelo no es capaz de entender ciertas expresiones como "hijos de su madre" o "subespecie" como odio, esto seguramente se debe a la falta de datos con este tipo de lenguaje, el modelo no ha visto suficientes textos así como para haber encontrado el patrón.

### 8.2.9 Clasificación de plataforma

Para poder comprobar si los datos de Youtube y Twitter podían juntarse se pensó en crear un modelo clasificador de plataforma. De este modo se puede comprobar la hipótesis siguiente: si un modelo puede encontrar un patrón para separar los datos entre las plataformas significa que los datos son distintos, ya que se ha encontrado una manera de identificar la plataforma, de lo contrario, significará que no hay diferencia entre las plataformas, pudiendo juntar los datos.

La arquitectura que se diseñó sigue el siguiente esquema:

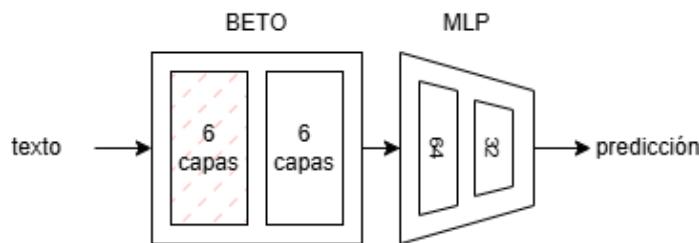


Ilustración 75: arquitectura modelo clasificador de plataforma

Se puede ver que tiene partes principales, un BETO y un MLP. Para hacer que el modelo entienda el contexto lo mejor posible se ha decidido entrenar las seis últimas capas del BETO, de este modo se consiguen embeddings adatados.

Los resultados son los siguientes:

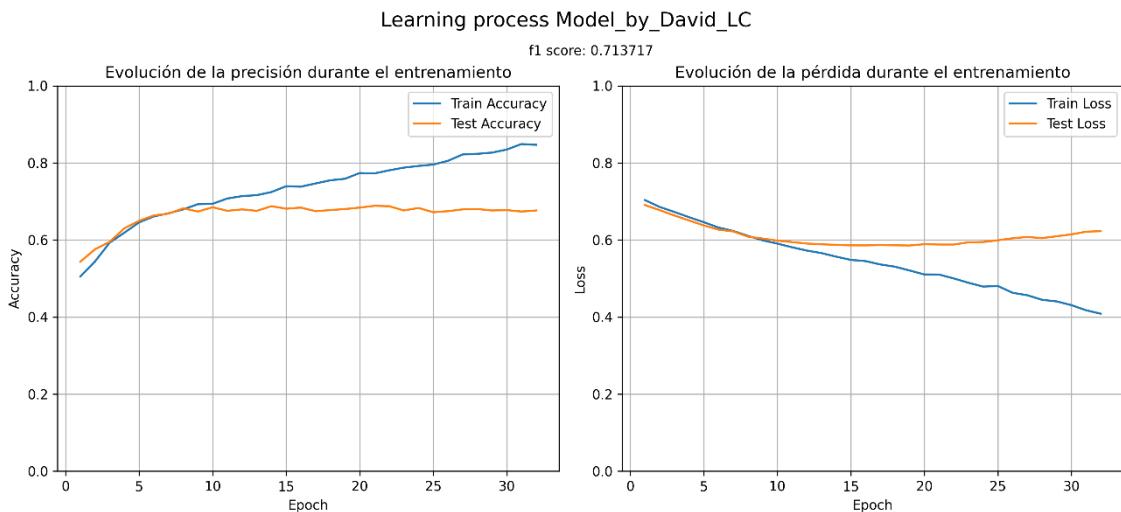


Ilustración 76: entrenamiento de clasificador de plataforma

Se puede ver que el aprendizaje del modelo se estanca a partir del *epoch* 60, el motivo por el cual no se activa el *early stopping* antes es porque, si se mira con detalle, el modelo tiene un aprendizaje bastante irregular, haciendo que no se active.

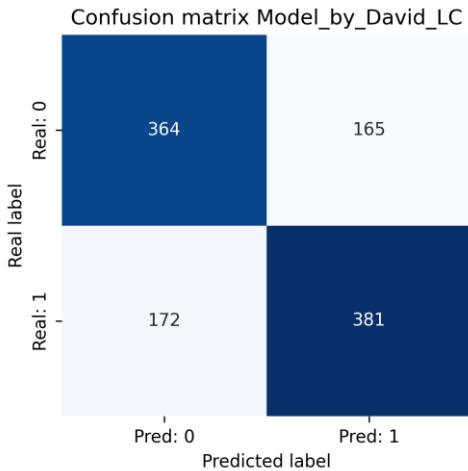


Ilustración 77: matriz de confusión del clasificador de plataforma

Se puede ver que la matriz resultante del modelo clasifica el odio y el no odio de manera bastante balanceada. Si se analizan las métricas del modelo se puede ver:

Tabla 17: resultados del modelo clasificador de plataforma

F1 score	Accuracy	Precision	Recall	ROC-AUC
$0.7659 \pm 0.1252$	$0.7660 \pm 0.1260$	$0.7794 \pm 0.0414$	$0.7562 \pm 0.6967$	$0.7662 \pm 0.1193$

Se puede ver que el modelo consigue una f1 de 0.7659, si se analiza el valor del *recall* (0.7562) se puede ver que está a la par que al valor de *precision* (0.7794) y el de f1 (0.7659), indicando que el modelo no favorece ninguna clase. Se puede confirmar que el modelo es capaz de discriminar correctamente las dos calases gracias al valor ROC-AUC (0.7662).

El modelo no puede separar los datos con la consistencia esperada como para poder confirmar la hipótesis planteada. No se ha encontrado una manera clara de separar los datos o de juntarlos como para que se pueda decir que las plataformas tienen el mismo comportamiento o que tienen comportamientos completamente distintos. El resultado indica que hay comentarios que sí que se pueden separar por plataforma, pero que muchos no tienen comportamientos significativamente distintos como para separarse por plataforma.

Al analizar los datos mal clasificados se pudo encontrar tweets como los siguientes:

- "¿Va a votar a comunistas porque en VOX no hay liberales? Pero que coj..."
- Clasificado como YouTube, pero proveniente de Twitter
- "Españoles no permitan que nadie llamé presidente a su corrupto.um"
- Clasificado como Twitter, pero proveniente de YouTube

- “Pide que vuelvan los liberales a Vox pero va a votar al Frente obrero que es antiliberal. 😂😂”
  - Clasificado como Twitter, pero proveniente de YouTube
- "@JLSanchez78 @mariohezonja @RMBaloncesto Por favor no nos hagas reír 🙄🙄"
  - Clasificado como Twitter, pero proveniente de YouTube

Si se analizan las palabras usadas en estos tweets de ejemplo mal clasificados, se puede ver que las palabras que tienen no pertenecen a ninguna claramente a ninguna plataforma (como se ha podido ver en el EDA). Si se analizan los emojis, se puede ver que la cantidad de comentarios con emojis es alta, esto se debe a que no se están usando para la clasificación.

Adicionalmente a este modelo, también se creó uno que usaba las puntuaciones de los emojis (se decidió usar las puntuaciones ya que previamente se había visto que los usuarios usan ciertos emojis más en Twitter que en Youtube y viceversa). Este modelo nuevo tiene una arquitectura muy similar a “RNN + MLP” (en cuanto al uso de la puntuación de los emojis). Tiene dos diferencias principales respecto “RNN + MLP”, no usa una RNN para procesar los embeddings de BETO y se entrena 6 de las 12 capas de BETO en vez de solo 1.

Los resultados de este modelo son prácticamente iguales a las anteriores, por lo que no se ha considerado necesario comentar la matriz de confusión y el proceso de entrenamiento por separado. Las métricas obtenidas son:

*Tabla 18: resultados del modelo clasificador de plataforma con emojis*

F1 score	Accuracy	Precision	Recall	ROC-AUC
0.7839 ± 0.0826	0.7840 ± 0.0814	0.8101 ± 0.0865	0.7550 ± 0.5381	0.7847 ± 0.0773

Se puede ver que las métricas han mejorado un poco respecto al modelo anterior, esto se debe a que, como se ha visto en el apartado de análisis de los datos, los emojis se usan distinto dependiendo de la plataforma. El poco incremento de los resultados seguramente se debe a la baja cantidad de emojis en el dataset.

Al mirar los comentarios mal clasificados se puede ver que el total de comentarios que tienen emojis ha bajado respecto el modelo anterior.

Aún que el modelo haya conseguido mejores resultados que el anterior no son suficientemente buenos como para indicar que se pueden combinar los datasets.

## 8.3 Replicación de estudios

Vista la influencia de los emojis en el procesado de los mensajes para el estudio del odio se pensó que sería buena idea intentar replicar proyectos hechos por otros equipos de investigación para así poder aprender la influencia de las distintas partes de cada elemento del estudio y así poder crear un modelo propio combinando las conclusiones de los trabajos.

Para poder hacer un estudio justo se ha visto la necesidad de crear un ambiente de evaluación de cada uno de los estudios, haciendo que las pruebas de los estudios se hagan siempre con los mismos datos, procesado exactamente igual, en serie y, lo más importante, de manera secuencial para que así se puede dejar el programa en ejecución durante largos períodos de tiempo.

### 8.3.1 Entorno de replicación

El ordenador usado es GeminAI, propiedad de la Salle. Las especificaciones son las siguientes:

- CPU: Intel Core i9-13900k de 13a generación
- GPUs : dos Nvidia RTX 4090
- RAM: 64 GB

El acceso a este ordenador se hace de manera remota y no tiene interfaz gráfica, creando la necesidad de realizar un informe visual en el proceso de replicación de experimentos. Este informe tiene que contener todos los pasos del pipeline de datos y de modelos, para así poder ver los pasos que se han hecho exactamente y poder identificar algún error si es que lo hay.

El entorno de evaluación de experimentos replicados se puede ver en el siguiente diagrama:

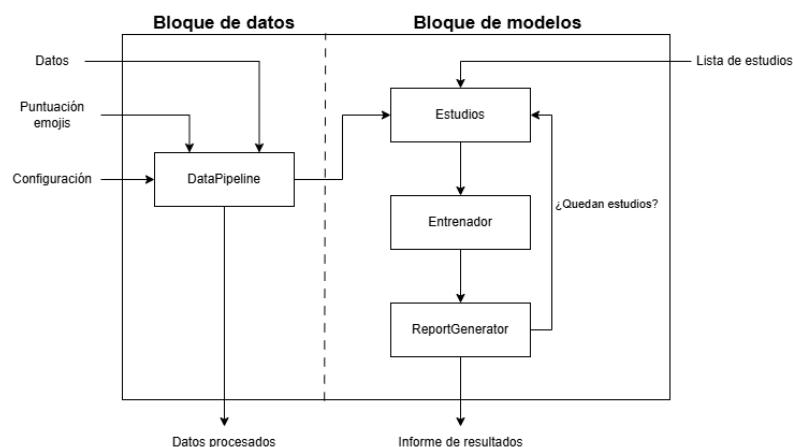


Ilustración 78: Diagrama pipeline datos y modelos

En el diagrama se puede ver como el sistema recibe cuatro inputs, datos, valoración de los emojis, configuración del procesado de los datos y lista de estudios.

El input de datos es el fichero que contiene los datos a procesar. La configuración del procesado de datos consiste en cómo se tiene que hacer la limpieza de datos, como se gestionan los emojis y como se crean los embeddings.

Una vez procesados los datos, se guardan en el sistema y se empieza la fase de entrenamiento de modelos. El entrenamiento de modelos (que se explicará en detalle más adelante), recibe una lista de experimentos a realizar y la configuración de estos, una vez terminados todos los experimentos se crea un informe de los resultados.

La programación de este apartado se ha hecho para maximizar la escalabilidad y la distribución de las responsabilidades de cada clase utilizando los principios SOLID.

Los principios SOLID fueron introducidos por Robert C. Martin en el paper “Desing Principles and Design Patterns”<sup>50</sup>, Michael Feathers ideó las siglas SOLID a partir de los resultados del paper de Robert C. Martin. Consisten en una serie de principios aplicados a la programación orientada a objetos (POO) para poder tener un código bien estructurado, escalable y claro.

El paradigma SOLID contempla los siguientes puntos de desarrollo:

- **S** (Single-responsibility Principle): cada clase solamente debe tener una única responsabilidad. Esto facilita la escalabilidad y facilita el mantenimiento.
- **O** (Open-closed Principle): las clases estar abiertas a extensión, pero no a modificación. Se debería poder añadir funcionalidades sin tener que modificar el código original (por ejemplo, mediante herencia).
- **L** (Liskov Substitution Principle): una clase debe poder substituir a su clase base (por su padre) sin alterar el comportamiento del programa. Una clase hijo no debe romper el comportamiento de una clase padre.
- **I** (Interface Segregation Principle): no se debe tener que obligar a una clase a implementar métodos que no necesita, se deben usar interfaces pequeñas para evitar sobrecargar las clases de responsabilidades que no necesitan.
- **D** (Dependency Inversion Principle): se debe utilizar abstracciones, no implementaciones concretas. Esto facilita el testeo y reutilización del código.

Al aplicar estos principios se ha conseguido un código modular, fácil de mantener y escalar que permite ejecutar múltiples experimentos de manera secuencial con distintas configuraciones, de este modo se puede recrear experimentos con las mismas condiciones de manera sencilla.

---

<sup>50</sup> (Martin, 2000)

A continuación, se va a ver el funcionamiento de los distintos bloques del entorno de replicación.

#### 8.3.1.1 Bloque de datos del pipeline

El pipeline de datos se encarga de procesar los datos y crear un fichero con los embeddings necesarios para el experimento. Los métodos de codificación implementados son los mismos que se han visto en el capítulo “Encodings y embeddings”.

El funcionamiento del módulo de datos es simple, solamente hace falta crear un objeto de clase *DataPipeline*, configurar sus atributos e iniciar el procesado de datos.

Los atributos que se pueden configurar son:

- Fichero mensajes: es la ruta del fichero con todos los mensajes sin procesar.
- Fichero puntuaciones emojis: es la ruta del fichero que contiene la información de los emojis (puntuación, tipo, nombre, etc.).
- *tokenize\_emojis*: esta variable indica si los emojis se deben convertir en tokens y dejar en el texto o si deben ser borrados.
- *codification\_type*: indica el tipo de codificación que se va a realizar. Puede ser de los tipos: “BERT”, “RoBERTa”, “BETO”, “One-hot”, “Tf-idf”, “Word2Vec” o “FastText”.
- *text\_name*: es el nombre de la columna que contiene los textos a analizar, de este modo se pueden intercambiar los datasets fácilmente.
- *label\_name*: es el nombre de la columna que contiene las variables target, de este modo se pueden intercambiar los datasets fácilmente.
- *hide\_prints*: durante el experimento se genera un informe que recolecta todos los pasos realizados. Estos pasos se guardan en un fichero y se muestran por la pantalla del terminal (el informe se puede ver en el capítulo “Informe final”). Este atributo permite al usuario esconder los pasos en el terminal.
- *show\_bert\_warnings*: BERT muestra mensajes durante la codificación de los embeddings. Esta variable permite verlos si se desea.

#### 8.3.1.2 Bloque de modelos del pipeline

Una vez procesados los datos, el pipeline continúa con la fase de modelos. Para que esta parte funcione solamente hace falta que se especifique el experimento a realizar y su configuración (más adelante se explica cómo hacer el proceso de creación de un experimento, en esta parte se explica su configuración). La configuración se especifica en una lista compuesta por diccionarios que sigue el siguiente formato:

- Título experimento: nombre del experimento que se mostrará en el informe final y del nombre de los ficheros que se generan.
- Modelo: clase que contiene la arquitectura del modelo del experimento.
- Parámetros del modelo: parámetros del modelo, que necesita para funcionar. Cada modelo puede tener atributos distintos, depende de su implementación.

- Función que ejecutar para entrenar: función de entrenamiento del modelo, se pueden usar las siguientes funciones:
  - *execute*: entrena el modelo con la configuración especificada.
  - *execute\_optimized*: entrena el modelo con optimización bayesiana para conseguir la mejor configuración posible.
- Parámetros de la función de entrenamiento, hace falta especificar los siguientes atributos:
  - *Dataloader* fase de entrenamiento
  - *Dataloader* fase de evaluación
  - Argumentos de los *dataloaders* usados en el experimento.

El diagrama de actividad que hace el pipeline de datos es el siguiente:

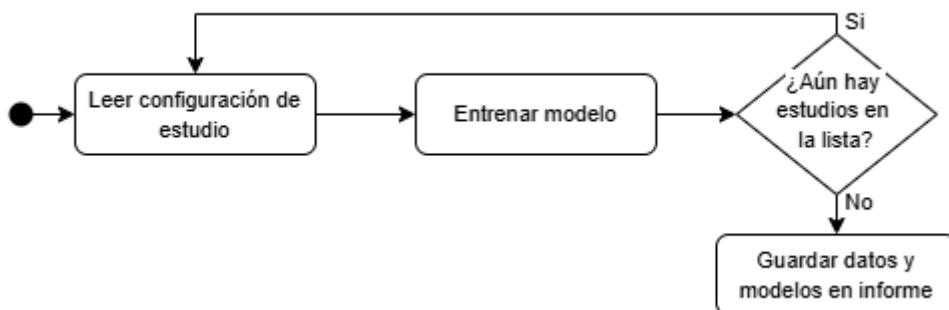


Ilustración 79: diagrama de actividad del pipeline de modelos

#### 8.3.1.3 Informe final

Durante el proceso de procesado de datos y de entrenamiento de los modelos en el pipeline se muestran los pasos realizados en la interfaz el terminal del ordenador y a la vez se guardan en un fichero informe. El fichero informe se guarda en una carpeta que se llama `./test_results/YYYY-mm-dd_HH-MM-SS_study`. En esta carpeta se encontrarán los siguientes elementos:

- *report.txt*: Fichero de texto plano que contiene el *output* del entrenamiento de los modelos.
- Ficheros del modelo: se genera un conjunto de elementos por cada estudio hecho.
  - *nombre\_modelo.pth*: fichero que contiene los pesos del modelo
  - *training\_process-nombre\_modelo.png*: fichero que contiene las gráficas con las estadísticas de entrenamiento del modelo.
- Ficheros de datos: carpeta que contiene todos los datos usados en los experimentos.
- Fichero de gráficas: carpeta que contiene todas las gráficas que han sido generadas.

A continuación, se puede ver un ejemplo del output de estudio:



Ilustración 80: menú principal del pipeline

Nota: el título de “Disargue” se ha generado con la página web ManyTools<sup>51</sup>, la fuente usada es “DOS Rebel”.

En esta parte del proceso de datos permite al usuario generar los datos o usar unos datos procesados previamente. Si quiere crear el fichero de datos se debe escribir “1”, si no se necesita crear un fichero de datos solamente se debe escribir “2”.

En el ejemplo actual se procesarán los datos.

---

<sup>51</sup> (ManyTools, 2025)

```
          DataPipeline

- hide_prints: False
- show_bert_warnings: False
- tokenize_emojis: True

Leyendo los datos
- Fichero './data/merged.xlsx' leido'
- Fichero './data/dataset_emoji_scores.csv' leido'

Creando dataset balanceado
- Tamaño dataset balanceado: 3894 (1947 datos por label)

Procesando texto

Ejecutando TextProcessor
- Con emojis tokenizados

Ejecutando EmojiProcessor
- Separando texto con emojis tokenizados
- Separando emojis
- Puntuando emojis

=====
BERT
=====

- device: cuda
- bert_type: dccuchile/bert-base-spanish-wwm-uncased
- save_file: ./save

- Procesando 500/3875 - 12.90%
- Procesando 1000/3853 - 25.95%
- Procesando 2000/3797 - 52.67%
- Procesando 2500/3775 - 66.23%
- Procesando 3000/3748 - 80.04%
- Procesando 3500/3716 - 94.19%
- Procesando 3711/3711 - 100.00%

- BERT Warning: el fichero './save.npz' ya existe
  Guardando fichero como './save_1747745338.npz'

- Fichero: './save_1747745338.npz' guardado

Ejecutando modelos:
- esBERT uncased with a batch size of 16
- esBERT uncased with a batch size of 32
- RoBERTa

Usando datos de: './save_1747745338.npz'
```

Ilustración 81: finalización del proceso de procesado de datos

Se puede ver que, en este caso, se ha usado BERT para conseguir los embeddings.

*Nota: en estas capturas solo se muestra el resultado del experimento “esBERT uncased with a batch size of 16”.*

```
esBERT
esBERT uncased with a batch size of 16

- dropout: 0.1
- device: cuda
- bert_model: dccuchile/bert-base-spanish-wwm-uncased

Batch arguments: ['input_ids', 'attention_mask']

=====
NN_trainer

- lr: 2e-05
- loss: CrossEntropyLoss()
- device: cuda
- scheduler: <function esBERT.execute.<locals>.<lambda> at 0x7feb4fb73d90>
- optimizer: <class 'torch.optim.adamw.AdamW'>

Training NN
- Epoch #1: f1_score 0.746433; accuracy 0.746619; avg_loss 0.49452887305191584
- Epoch #2: f1_score 0.761317; accuracy 0.761948; avg_loss 0.4968015150300094
- Epoch #3: f1_score 0.776623; accuracy 0.777277; avg_loss 0.6111917106168611

- Final F1: 0.7766229762998855

=====
Saving elements in folder: ./test_results/2025-05-20_14-48-46_study/
- Plots saved as: 'trainig_process-esBERT_uncased_with_a_batch_size_of_16.png'
- Model saved as: 'esBERT_uncased_with_a_batch_size_of_16.pth'
```

Ilustración 82: resultado del experimento esBERT

Una vez se han ejecutado todos los modelos se puede ver que se ha generado una carpeta de informe con todos los elementos mencionados previamente.

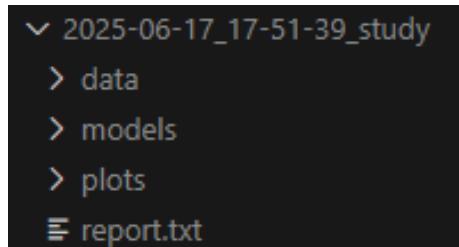
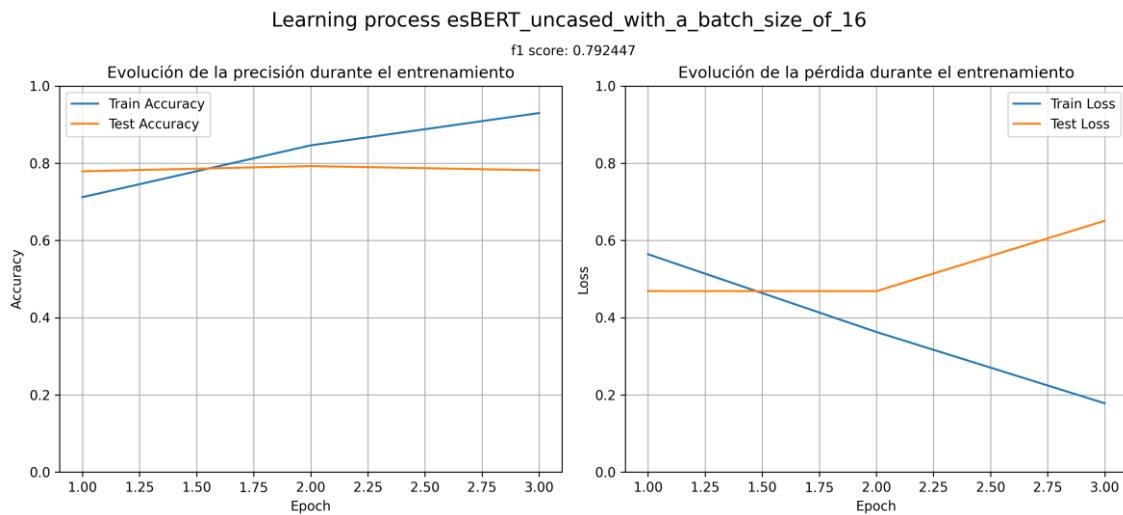


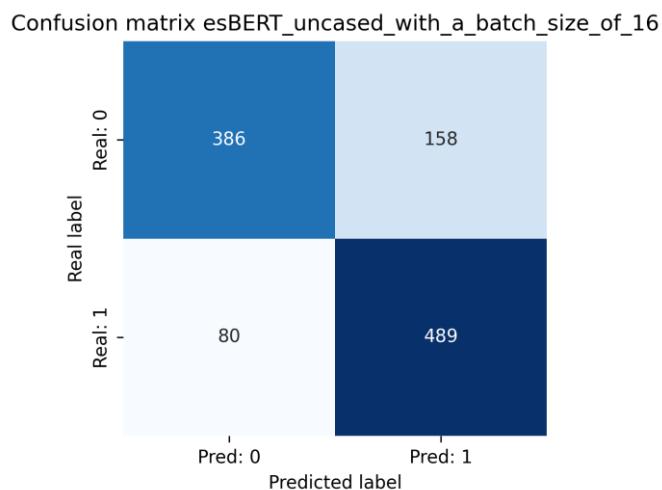
Ilustración 83: resultado del entrenamiento

Al abrir el fichero “traing\_process-esBERT\_uncased\_with\_a\_batch\_size\_of\_16.png” se puede ver la siguiente imagen:



*Ilustración 84: accuracy y pérdida del estudio esBERT uncased con batch size 16*

Si se mira el fichero “confusion\_matrix-esBERT\_uncased\_with\_a\_batch\_size\_of\_16.png”:



*Ilustración 85: matriz de confusión del estudio esBERT uncased con batch size 16*

### 8.3.1.3.1 Creación de estudios

En este apartado se va a ver cómo se pueden crear modelos personalizados. Dependiendo de si el modelo es de Machine Learning (ML) o de Deep Learning (DL) se debe hacer una implementación u otra.

Si el modelo es de ML se debe crear una clase que herede de la clase  *BaseModel*. Esta clase padre tiene los métodos:

- Funciones de entrenamiento: está el método *execute* (solamente ejecuta el modelo con la configuración que se especifique en el pipeline) y el método *execute\_optimized* (utiliza optimización bayesiana).

Si el modelo es de ML se debe crear una clase que herede de la clase  *BaseModel*. Esta clase padre tiene los métodos:

- Funciones de entrenamiento: está el método *execute* (solamente ejecuta el modelo con la configuración que se especifique en el pipeline) y el método *execute\_optimized* (utiliza optimización bayesiana).
- Funciones de gráficas: funciones que generan las gráficas con los resultados de los modelos.
- Función de comprobación de formato de los datos: esta función comprueba que los datos estén en el formato que necesita el modelo.
- Función de trials: es una función que debe implementar la clase hijo (es decir, la que especifica el modelo), devuelve los parámetros a optimizar con optimización bayesiana.

La clase hijo solamente debe implementar:

- El constructor: en el constructor se deben especificar los parámetros:
  - Modelo para ejecutar, por ejemplo: SVC, RandomForestClassifier, etc.
  - Dataset con los datos
  - Nombre de la fila con los embeddings
- Función de trials: esta función específica que parámetros se deben optimizar durante el proceso de optimización bayesiana. Si no se utiliza la funcionalidad de optimización bayesiana no hace falta hacer su implementación.

Es decir, para poder crear un experimento con un modelo de ML solamente hace falta programar su constructor y la función de trials (si se quiere hacer optimización bayesiana).

*Nota: todos los modelos explicados previamente (los del apartado de Machine learning (ML)) han sido replicados para poder funcionar en el pipeline.*

Para poder crear un modelo de DL se debe crear una clase hija de *NN\_arquitecture*.

La clase padre funciona igual que *BaseModel*, solamente se debe ejecutar la función *execute* para empezar el entrenamiento. Es necesario implementar el método *get\_default\_trainer*, este método devuelve una instancia inicializada de *NN\_trainer*, especificando el optimizador, scheduler, learning rate y loss function.

Las funciones de entrenamiento, evaluación y guardado de las redes neuronales se encuentran en *NN\_trainer*. Las clases implementadas en *NN\_trainer* son:

- *Training\_loop*: función que realiza todo el entrenamiento de la red.
- *Train*: función que se encarga de la etapa de entrenamiento, se usa en cada *epoch*. Entrena con los *epochs* del *dataloader* que se le entreguen.
- *Evaluate*: función que se encarga de la etapa de evaluación, se usa en cada *epoch*.
- *Test*: función que se encarga de la etapa final de evaluación, solamente se usa al terminar el entrenamiento.
- *Examineate*: esta función que evalúa al modelo con los *batches* que se le indiquen.
- Funciones de guardado de modelo y de gráficas de resultados y del proceso de entrenamiento del modelo

*Nota: todos los modelos explicados previamente (los del apartado Deep Learning (DL)) han sido replicados para poder funcionar en el pipeline de datos.*

### 8.3.2 Papers replicados

En este apartado se replican distintos artículos científicos para poder comprobar sus resultados. Estos estudios se replicarán en el entorno de replicación explicado en el apartado anterior con la intención de investigar si los modelos usados consiguen buenos resultados con los datos actuales.

#### 8.3.2.1 esBERT

El artículo “Spanish pre-trained BERT model and evaluation data”<sup>52</sup> se pretende encontrar un modelo de BERT en español con la intención de crear una arquitectura capaz de realizar tareas de clasificación con textos en español. Para realizar este estudio hacen múltiples pruebas para evaluar el modelo, entre ellas clasificación de textos mediante un MLP.

Para replicar el experimento correctamente se ha hecho *fine tuning* del modelo con 3 epochs, 10% de los steps de warm-up, usando el optimizador de AdamW y el shceduler lineal (como se especifica en el artículo). También se ha añadido una capa de clasificación en el modelo. La arquitectura se puede ver en la siguiente imagen:

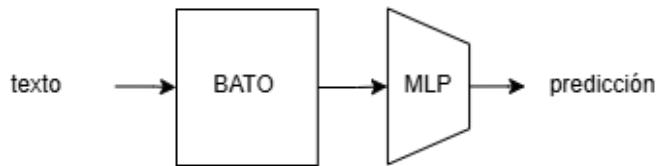


Ilustración 86: arquitectura del modelo del artículo esBERT

Los resultados son los siguientes:

Tabla 19: resultados del modelo esBERT en los distintos datasets

Dataset	Dataset	F1 score	Accuracy	Precision	Recall
Youtube (emojis tokenizados)	0.6918 ± 0.2822	0.6939 ± 0.2198	0.7003 ± 1.3843	0.7004 ± 7.1344	0.6938 ± 0.2177
Youtube (sin emojis)	0.6784 ± 0.3669	0.6815 ± 0.2802	0.6823 ± 0.9060	0.6956 ± 11.9499	0.6815 ± 0.2773
Twitter (emojis tokenizados)	0.7781 ± 0.0857	0.7781 ± 0.0857	0.7781 ± 0.0857	0.7781 ± 0.0857	0.7781 ± 0.0857
Twitter (sin emojis)	0.7592 ± 0.1715	0.7602 ± 0.1504	0.7612 ± 0.7015	0.7841 ± 1.5412	0.7594 ± 0.1668

<sup>52</sup> (José Cañete, 2020)

Todo (emojis tokenizados)	<b>0.7837 ± 0.1411</b>	<b>0.7842 ± 0.1421</b>	<b>0.7816 ± 0.3544</b>	<b>0.8027 ± 2.0176</b>	<b>0.7838 ± 0.1343</b>
Todo (sin emojis)	0.7781 ± 0.0857				

Nota: las varianzas están en escala  $10^{-3}$

Se puede ver que el dataset que ofrece mejores resultados es el de los datos de YouTube y Twitter combinados, el dataset de Twitter consigue valores muy parecidos, indicando que la diferencia entre los datasets, en este caso, no es muy significativa. El uso de los emojis tokenizados ha mostrado un incremento en los resultados obtenidos por ambos datasets.

Si se analizan los comentarios mal clasificados del dataset de los datos combinados, se puede ver:

- “@jin87mugen @MartaMonforteJ @LaHoraTVE Los voxeros sois más de turriar”
  - Clasificado como odio, etiquetado como no odio
- “@MonicaCarrillo Buenos días Mónica”
  - Clasificado como no odio, etiquetado como odio
- “@AlmudenaMF No invente, señora.”
  - Clasificado como no odio, etiquetado como odio

Los resultados mal clasificados tienen textos que contienen ironía u odio no implícito (el modelo requeriría de más contexto para detectarlo). También se puede ver que muchos de los textos mal clasificados tienen palabras o insultos poco comunes, esto confunde al modelo.

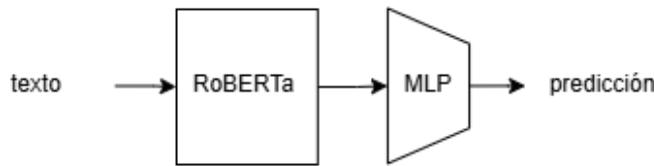
### 8.3.2.2 RoBERTa

En el artículo “RoBERTa: A Robustly Optimized BERT Pretraining Approach”<sup>53</sup> describe el proceso de creación de la variante de BERT, RoBERTa. Entre las diversas evaluaciones que realizan, incluyen tareas de clasificación para evaluar su rendimiento. En el artículo se detalla las modificaciones hechas en el entrenamiento de BERT para crear a RoBERTa, estos son: aplicación de *masking* dinámico, eliminan el *next sentence prediction*, modifican el tamaño de los *batches* y utilizan más datos (usan frases más largas y aumentan el vocabulario).

En este experimento se ha decidido utilizar la misma arquitectura que en el capítulo anterior, pero esta vez utilizando a RoBERTa en vez de a BERT. La arquitectura usada es la siguiente:

---

<sup>53</sup> (Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov, 2019)

*Ilustración 87: arquitectura del modelo del artículo RoBERTa*

Los resultados obtenidos son los siguientes:

*Tabla 20: resultados del modelo RoBERTa en los distintos datasets*

<b>Dataset</b>	<b>Dataset</b>	<b>F1 score</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>
Youtube (emojis tokenizados)	0.6427 ± 0.1265	0.6445 ± 0.1200	0.6507 ± 0.4039	0.6351 ± 0.5947	0.6446 ± 0.1186
Youtube (sin emojis)	0.6405 ± 0.1155	0.6443 ± 0.0863	0.6373 ± 0.5036	0.6970 ± 0.7899	0.6438 ± 0.0855
Twitter (emojis tokenizados)	<b>0.7649 ± 0.4165</b>	<b>0.7983 ± 8.6396</b>	<b>0.7806 ± 1.9994</b>	<b>0.7691 ± 3.7969</b>	<b>0.7661 ± 0.3970</b>
Twitter (sin emojis)	0.7255 ± 0.1871	0.7278 ± 0.1477	0.7184 ± 0.4455	0.7837 ± 0.6432	0.7257 ± 0.1708
Todo (emojis tokenizados)	<b>0.7681 ± 0.1721</b>	<b>0.7684 ± 0.1674</b>	<b>0.7851 ± 0.3068</b>	<b>0.7512 ± 2.0481</b>	<b>0.7687 ± 0.1590</b>
Todo (sin emojis)	0.7387 ± 0.5689	0.7397 ± 0.5135	0.7601 ± 0.2115	0.7239 ± 0.9238	0.7401 ± 0.4405

Nota: las varianzas están en escala  $10^{-3}$

Al igual que con el modelo anterior, se puede ver que el dataset de datos de Twitter y el de comentarios de YouTube y Twitter consiguen resultados parecidos. El modelo no favorece ninguna clase en particular, es estable. Al analizar los textos mal clasificados se puede ver comentarios como:

- "Increíble se trabaja los lunes en andorra"
  - Clasificado como no odio, etiquetado como odio
- "@rebecacrespo\_ Que va a decir una Voxera 🤣🤣🤣🤣 se os ha acabado el chollo bonicos!!!"
  - Clasificado como no odio, etiquetado como odio
- "Vamooooossss Le Pen!!!! También despierta Francia!!! Todos menos España. Lamentable!!!! Desde Cádiz ESPAÑA"
  - Clasificado como odio, etiquetado como no odio

Analizando los datos mal clasificados se puede ver que el uso de la ironía y de palabras poco comunes es habitual. El modelo tampoco puede detectar el sentimiento correctamente cuando los emojis y el texto transmiten sentimientos opuestos.

## 9 Etiquetaje de tweets mediante LLM

Después de haber probado todos los modelos explicados anteriormente y de haber investigado los errores que cometían los modelos se confirmó la hipótesis de que probablemente el etiquetaje de los tweets no es consistente.

Los motivos principales para llegar a esta conclusión son los siguientes:

1. El proceso de etiquetación de los datos por parte de la Universidad de Blanquerna se ha hecho con múltiples etiquetadores, cada etiquetador se ha encargado de una parte del total de los datos. Esto puede hacer que los etiquetajes no sean consistentes.
2. En otro proyecto parecidos, en el que se usan los mismos datos, se ha encontrado que los modelos solamente aprenden hasta cierto punto (no se consigue superar el valor de la f1 0.8 de ninguna de las maneras), esto parece indicar que hay un porcentaje de los datos que no están bien etiquetados, por eso ningún modelo consigue mejorar.  
Al analizar los datos que los modelos de estos otros proyectos se ha encontrado que hay tweets clasificados de manera poco clara.
3. En un proyecto de detección de toxicidad, pero esta vez enfocado a las mujeres, han conseguido mejorar el etiquetaje mediante *Large Language Models* (LLM), aparentemente solucionando el problema de los etiquetajes poco consistentes.

Investigando el estado del arte sobre el etiquetaje de odio mediante técnicas automáticas se decidió que una posible manera de solucionar el problema consiste en utilizar *Large Language Models* (LLM).

Un LLM es un modelo de aprendizaje automático entrenado sobre grandes cantidades de texto. Escriben textos intentando adivinar la siguiente palabra mediante el cálculo de la probabilidad de que una palabra aparezca dadas las palabras anteriores.

Por ejemplo, si tenemos la frase: “El máster del MUDS ha sido ...”, la probabilidad de que la siguiente palabra sea “manzana”, “proceso”, “planeta” o “María” son muy bajas, en cambio la probabilidad de que sea “interesante”, “difícil”, “largo” o “doloroso” tienen probabilidades más altas ya que tienen más sentido que los primeros ejemplos mostrados.

Al tener esta capacidad de predicción se puede hacer que un LLM haga tareas como: generación, resumen, clasificación o traducción de textos, generar código o responder preguntas sobre el texto.

Hay muchos tipos de LLM, para elegir uno para satisfacer las necesidades del problema se deben tener en cuenta las siguientes características:

- Nombre de parámetros: son los valores que ha aprendido el LLM durante el entrenamiento, son como los pesos de una red neuronal.

Un modelo como GPT-2 tiene 124M de parámetros en la versión más simple y 1.5B de parámetros en el modelo más complejo. El modelo de Meta, LLaMA 2 está disponible en las versiones de 7B, 13B i 65B de parámetros.

Tener un modelo con más parámetros tiene las ventajas de que tiene la información mejor representada (entiende mejor los datos), captura mejor las relaciones semánticas y sintácticas y tiene mejor precisión para tareas más complejas. Los inconvenientes de un modelo grande es que, al ser más complejo, requiere de más capacidad de computación, ocupa más espacio que uno de más simple y tiene más facilidad para sobreajustarse a una tarea y no saber generalizar correctamente.

- Arquitectura del *transformer*: los LLM están formados por *Transformers*. La arquitectura de los Transformers determina características como:
  - Capacidad de atención: es la capacidad del modelo de medir como de relevante son las palabras entre sí.
  - Capacidad de parallelización: es la capacidad de interpretar las palabras simultáneamente en vez de tener que hacerlo secuencialmente como en las *Recurrent Neural Networks* (RNN).
  - Capas: cantidad de capas que transforman la representación del texto de manera abstracta.
- Ventana de contexto: capacidad que tiene el modelo de “ver” o procesar texto a la vez, es decir, capacidad que tiene el modelo de retener información “en mente” para entender el significado de una palabra y así poder predecir la siguiente. Esta característica es importante para:
  - Comprender el significado de una palabra en relación con el contexto.
  - Mantener la coherencia durante la respuesta.
- Métricas de evaluación: una característica para tener en cuenta al elegir cualquier modelo es las métricas que ha conseguido en las pruebas que se le han hecho.
  - F1 score, precisión, robustez, etc.

Para introducir una instrucción a un LLM se tiene que escribir en un texto lo que se necesita que realice, esta instrucción se llama *prompt*.

Para poder hacer el *prompt* que indique al LLM como debe hacer la clasificación se debe especificar los criterios que determinan si un texto contiene toxicidad o no. Para poder hacer una comparación justa entre etiquetadores hace se tienen que usar los mismos criterios de clasificación. Los criterios de clasificación se encuentran en el capítulo “Criterios de clasificación de Blanquerna”.

El *prompt* que se ha utilizado teniendo en cuenta los criterios de Blanquerna es el siguiente:

Tabla 21: *prompt con los criterios de Blanquerna*

Eres un sistema automático para detectar ataques online a periodistas.  
Lee el siguiente tweet y responde solo con '1' si el tweet es un ataque según estos criterios:

- Incivismo, insultos o abusos verbales ("Eres un pésimo periodista", "Eres un cabrón")
- Amenazas o intimidación explícita o implícita ("Te pasará algo", "Te enseñarán una lección")
- Discurso de odio por género, raza, ideología, LGTBQ+, etc.
- Comentarios sobre la apariencia física (no relacionados con el trabajo) ("Eres muy guapa", "Me encantaría estar contigo")

Responde '0' si el tweet no entra en ninguna de estas categorías (simple opinión, crítica profesional, etc.).

Responde solo con el número '0' o '1'. No añadas nada más.

Para comprobar la hipótesis de que un LLM puede hacer bien el etiquetaje se ha decidido entrenar a tres modelos especializados en detección de odio y tres genéricos usando el *prompt* anterior, finalmente se va a hacer un *test* de *Kendall* para comparar la consistencia de los etiquetadores.

## 9.1 Capa de Kendall

El *test* de *Kendall*<sup>54</sup>, también llamado coeficiente Tau ( $\tau$ ), es una prueba estadística que se usa para medir si hay relación entre dos variables. En el contexto de anotación de datos, sirve para comprobar si los etiquetadores están siguiendo el mismo criterio.

Antes de entrar en detalle con el *test* de la capa de *Kendall* hace falta recordar que es el *p-value*. El *p-value*<sup>55</sup> indica la probabilidad de obtener un resultado igual o más extremo que el observado asumiendo que se cumple la hipótesis nula ( $H_0$ ).

Las hipótesis planteadas en el *test* de *Kendall* son las siguientes:

- $H_0$ : no hay relación entre los etiquetadores
- $H_1$ : hay relación entre los etiquetadores

---

<sup>54</sup> (Kendall, 1938)

<sup>55</sup> (DataTab, 2025)

Si el valor p obtenido es menor que 0.05 se rechaza  $H_0$ . En el test de *Kendall* esto significa que existe una relación estadísticamente relevante entre los etiquetadores.

Volviendo al concepto de test de *Kendall*, calcula la similitud entre dos variables. Este valor es tau ( $\tau$ ). Este valor está entre el 1 y el -1, tiene el siguiente significado:

- $\tau = 0$ : las variables no tienen relación entre sí.
- $\tau > 0$ : relación directa. Cuanto mayor  $\tau$ , mayor concordancia.
- $\tau < 0$ : relación inversa. Cuanto más negativo  $\tau$ , mayor discordancia.

## 9.2 Comparativa de etiquetaje de LLM

A continuación, se pueden ver dos tablas con los resultados obtenidos con los modelos seleccionados. La primera tabla contiene los resultados de LLM especializados con detección de odio, la segunda tabla contiene los resultados de LLM generativos a los que se les ha introducido el *prompt* anterior para que generen la respuesta.

Cada tabla tiene el nombre del LLM, el valor de tau conseguido en el test de *Kendall* al compararlo con el etiquetaje original, el p-value conseguido y el valor de la métrica f1 conseguido en el modelo RNN + MLP (el del capítulo “RNN + MLP”).

El motivo por el que se ha elegido el modelo RNN + MLP es porque es el que ha conseguido mejores resultados (el modelo RNN + MLP con emojis codificados con one-hot encoding consigue resultados prácticamente iguales, pero necesita más tiempo de entrenamiento).

*Tabla 22: resultados LLM especializados en detección de odio*

LLM especializados	$\tau$	p-value	F1 de RNN + MLP
HateSpeech-BETO-cased-v2 <sup>56</sup>	0.21827	7.4462e-40	0.7878 ± 0.3375
mrm8488/bert-tiny-finetuned-sms-spam-detection <sup>57</sup>	0.05364	0.00116738	0.8574 ± 0.572
pysentimiento/robertuito-hate-speech <sup>58</sup>	0.17438	4.7886e-26	0.7845 ± 0.6162

*Nota: las varianzas están en escala 10<sup>-3</sup>*

---

<sup>56</sup> (Sánchez, 2025)

<sup>57</sup> (mrm8488, 2025)

<sup>58</sup> (pysentimiento, 2025)

Tabla 23: resultados LLM generativos

LLM generales	$\tau$	p-value	F1 de RNN + MLP
Qwen/Qwen1.5-1.8B-Chat	-0.0106	0.5180	$0.6231 \pm 0.504$
meta-llama/Meta-Llama-3-8B-Instruct <sup>59</sup>	0.36233	1.2717e-106	$0.7940 \pm 0.294$

Nota: las varianzas están en escala  $10^{-3}$

Se puede ver que todos los coeficientes de *Kendall* son bajos (al compararlos con los etiquetajes de Blanquerna deberían tener un valor más cercano a 1). El *p-value* indica que confirma que el resultado de la correlación es estadísticamente relativo, indicando que la correlación es débil.

Esto sorprende ya que, el modelo *mrm8488/bert-tiny-finetuned-sms-spam-detection* ha conseguido un valor de f1 de 0.8497, superando el límite de 0.8 que se había encontrado previamente. Al analizar los otros parámetros que consigue el modelo RNN+MLP con este dataset se puede ver los siguiente:

Tabla 24: resultados de RNN+MLP en dataset mrm8488

F1 score	Accuracy	Precision	Recall	ROC-AUC
$0.8574 \pm 0.5717$	$0.8478 \pm 0.5584$	$0.8679 \pm 0.4530$	$0.8433 \pm 0.00467$	$0.8577 \pm 0.5717$

Nota: las varianzas están en escala  $10^{-3}$

Viendo los resultados de la tabla anterior se puede concluir que el modelo ha aprendido bien, el *recall* (0.8433) no es muy distinto a la precisión (0.8679) ni al valor f1 (0.8574), si se mira el valor ROC-AUC se pude ver que es de 0.8577, confirmando que el modelo está aprendiendo bien. Esto demuestra que el modelo de mrm8488 puede clasificar el odio con un criterio más consistente al del dataset original.

Si se analizan los otros resultados de los otros modelos, se puede ver que son prácticamente iguales a los obtenidos al usar el dataset original. Esto significa que se ha encontrado una manera distinta de detectar la toxicidad a la original.

---

<sup>59</sup> (Meta, 2025)

## 9.3 Evaluación LLM

Visto que los resultados son buenos se decidió hacer un estudio más exhaustivo de los datos. Para ello se decidió etiquetar manualmente un fragmento de 200 comentarios aleatorios, 100 clasificados como odio y 100 como no odio. De este modo se puede estar seguro de que el criterio de etiquetaje se haya hecho por la misma persona.

El motivo por el cual no se puede hacer esta prueba con los datos originales es porque en el dataset no contiene información de los etiquetadores, haciendo imposible separar los comentarios por etiquetador.

Una vez etiquetados los datos se hará una comparativa igual que en el capítulo anterior, es decir se usarán los LLM para etiquetar los comentarios y se usará el test de Kendall para comprobar si hay uniformidad en el etiquetaje. Finalmente se comprobarán los resultados de los etiquetajes con el modelo RNN + MLP.

Los resultados conseguidos por el modelo RNN + MLP con el dataset etiquetado manualmente son los siguientes:

- F1 score:  $0.7761 \pm 0.1289$
- Accuracy:  $0.78 \pm 0.12$
- Loss:  $0.5398 \pm 0.3601$

A continuación, se va a probar de usar los mismos LLM usados en el capítulo anterior para que etiqueten el dataset etiquetado manualmente.

*Tabla 25: resultados LLM especializados en detección de odio*

LLM especializados	$\tau$	p-value	F1 de RNN + MLP
HateSpeech-BETO-cased-v2	0.18082	7.0058e-28	$0.5685 \pm 0.26$
mrm8488/bert-tiny-finetuned-sms-spam-detection	0.05364	0.00116738	$0.4977 \pm 0.3$
pysentimiento/robertuito-hate-speech	0.23719	0.00084566	$0.5164 \pm 0.28$

Nota: las varianzas NO están en escala  $10^{-3}$

Se puede ver que los resultados de la capa de *Kendall* no son muy buenos, los etiquetajes no se parecen mucho a las etiquetas manuales.

Si se comparan los resultados del modelo RNN + MLP entre los datos etiquetados de los LLM y manualmente se puede ver que los resultados de los datos etiquetados manualmente tienen mucha menos varianza que los datos de los LLM. Esto es normal ya que el dataset manual tiene 200 datos, donde cada clase tiene una representación del 50%, en los otros datasets esa proporción no es del 50%, haciendo que la cantidad de datos se reduzca mucho más. Al tener tan pocos datos el modelo no acaba de aprender bien, haciendo que una pequeña diferencia entre los datos usados para entrenamiento o evaluación hagan que el modelo consiga métricas distintas.

Si miramos los resultados de los LLM genéricos:

*Tabla 26: resultados LLM generativos*

LLM generales	$\tau$	p-value	F1 de RNN + MLP
Qwen/Qwen1.5-1.8B-Chat	0.04122	0.0125717	$0.7861 \pm 0.0905$
meta-llama/Meta-Llama-3-8B-Instruct	0.49813	2.39696e-12	$0.8029 \pm 0.0338$

Nota: las varianzas NO están en escala  $10^{-3}$

Al igual que con los modelos especializados en el odio, los valores tau de estos modelos de lenguaje no son suficientemente altos como para indicar que se ha seguido el mismo criterio de etiquetaje. También se ha podido ver que los resultados del modelo RNN + MLP con estos datos es muy parecido a los datos originales, indicando que se ha encontrado un patrón de comportamiento del odio distinto al especificado por Blanquerna.

En conclusión, se ha conseguido hacer que un LLM etiquete los datos de manera más consistente que con los etiquetadores humanos. Esto ha incrementado los resultados que se obtenían con los modelos en los datos originales, se ha conseguido superar exitosamente el límite de f1 de 0.8.

Este hallazgo demuestra la importancia que tienen los criterios de etiquetaje consistentes al clasificar datos. La metodología usada en los datos originales fue la siguiente: se escribieron unas pautas para indicar que se considera odio o no, se separaron los datos en n grupos y se pidió a n personas que lo clasificaran. Esto resultó en un dataset con un etiquetaje inconsistente, empeorando el proceso de aprendizaje de los modelos. Al usar un LLM se ha podido garantizar que el criterio de clasificación sea el mismo, consiguiendo mejorar los resultados. El motivo por el cual el modelo RNN+MLP no consigue mejorar los resultados aún más de debe, seguramente, a la falta de datos, haciendo que el modelo no consiga encontrar algunos patrones de toxicidad poco usados o muy específicos.

# 10 Coste temporal y económico

En este capítulo se detalla el coste temporal de dedicación al proyecto (contando el desarrollo y la redacción de la documentación) y el coste económico de haber realizado el proyecto.

## 10.1 Coste temporal

La dedicación semanal de este proyecto ha sido de aproximadamente 35 horas por semana, esto da un total de una dedicación de 560 horas.

A continuación, se muestra en un diagrama de *Gantt* la distribución de cada parte del proyecto:

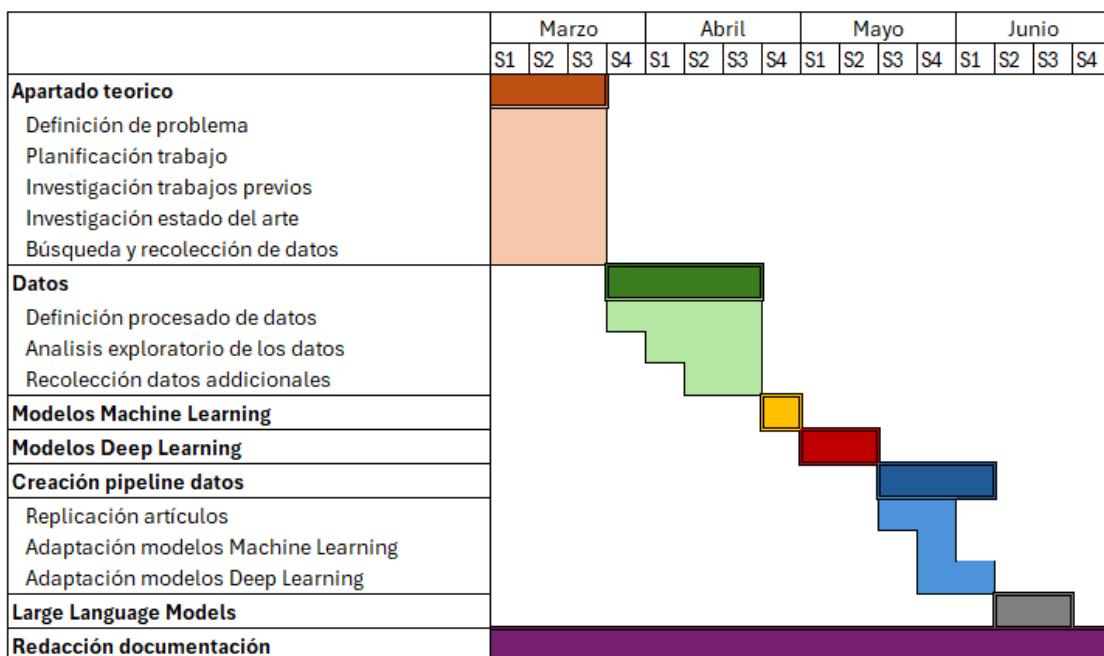


Ilustración 88: diagrama de Gantt

Como se puede ver, se han separado las fases del proyecto por colores para facilitar la interpretación del diagrama. Las primeras tres semanas fueron principalmente dedicadas a la comprensión del problema de forma teórica y a planificar el enfoque del proyecto mediante el estudio del estado del arte. Las siguientes cuatro semanas se centraron en la recolección, definición e interpretación de los datos. A continuación, se empezaron las fases de desarrollo de modelos (la zona amarilla y zona roja) y del pipeline de datos (zona azul). Finalmente se desarrolló la parte de los LLM (zona gris). La redacción de los progresos conseguidos se realizó a lo largo de todo el desarrollo del proyecto.

## 10.2 Coste económico

Para el cálculo del coste económico del proyecto se han tenido en cuenta dos tipos de costes. El primero serían los gastos considerados como fijos y, segundo, los costes de infraestructuras usadas para el proyecto.

En cuanto a los costes fijos, se tendrá en cuenta el coste de contratación de un científico de datos con una dedicación de 30 horas semanales durante 4 meses. Para conseguir el salario medio actual de un científico de datos junior (entre 1 año y 3 de experiencia) en España se ha consultado *The Bridge*<sup>60</sup>. Según *The Bridge*, un salario para un trabajador de estas características oscila entre 30.000 y 40.000 euros brutos. El coste total que dedicaría una empresa para contratar a un científico de datos sería de 7.500€ (cálculo de salario mensual de jornada laboral reducida).

En cuanto a los costes de infraestructuras, para este proyecto se ha utilizado un ordenador de propiedad de LaSalle. Este ordenador se utiliza para diversos proyectos (no se ha comprado para realizar este proyecto). En el caso de que se quisiera realizar este proyecto en las mismas condiciones, se tendría que comprar un ordenador de las mismas características. Para calcular el precio del ordenador se ha utilizado *PcPartPicker*<sup>61</sup>, y el precio aproximado sería de aproximadamente 7700€. Las características del ordenador son:

- CPU: Intel Core i9-13900k de 13a generación. Precio: 385 €
- GPUs: dos Nvidia RTX 4090. Precio: 3179,99€ x 2 = 6359,98 €
- RAM: 64 GB. Precio: 234,99 €
- Placa madre: MSI B760. Precio: 169,99 €
- Refrigeración: Thermalright Peerless. Precio: 34,9 €
- Memoria: Samsung 990. Precio: 149,99 €
- Fuente de corriente: Corsair HX1500i. Precio: 264,99 €
- Caja: Corsair 4000D. Precio: 94,99 €

En el caso de que no querer invertir tal cantidad de dinero en la infraestructura, se podrían usar servicios de procesamiento de *cloud computing* como *Google Colab*, con un coste total de 11,19 € al mes. Esto subiría el precio total de infraestructura del proyecto hasta 44,76 €.

Contando el ordenador y el salario de contratación, se alcanzaría un precio total de proyecto de 15,700 €. Contando el salario de contratación y la suscripción del servicio *cloud*, el coste total de proyecto sería de aproximadamente 7.544,76 €.

---

<sup>60</sup> (*The Bridge*, 2024)

<sup>61</sup> (*PcPartPicker*, 2025)

## 11 Líneas futuras

En este apartado se indican líneas de trabajo que se consideran relevantes para poder continuar un trabajo de detección de odio.

1. Creación de un sistema de etiquetaje más consistente. Este apartado podría solventarse con dos posibles enfoques:

- a. Investigar cómo clasifican el odio los modelos de lenguaje.
  - b. Crear un sistema de etiquetaje formado por humanos que evalúen con criterios más estrictos.

El etiquetaje de los datos se vería altamente beneficiado por un sistema de comparación entre los etiquetadores para así poder garantizar la consistencia de los datos y que se sigan los mismos criterios de evaluación.

2. Investigar los perfiles “generadores de odio”. Detectar e investigar el odio en función de las relaciones que tienen los usuarios entre sí mediante la creación de grafos. Esto permitiría identificar comportamientos y comunidades entre los distintos usuarios.
3. Para mejorar los resultados de los modelos se podría crear un *Staking* que combine múltiples modelos especializados, para poder recoger más contexto y tener en cuenta más elementos aparte de emojis y texto. Los otros elementos o metadatos que se podrían tener en cuenta son: cantidad de mayúsculas (que indican si el usuario grita o no), relaciones con otros usuarios, comportamiento reciente en la red social, historial total del comportamiento en la red, etc.
4. Uso de técnicas de *Data Augmentation* (DA) para incrementar la cantidad y diversidad de datos. Aplicar distintas técnicas de DA puede permitir generar datos artificiales que ayuden a descubrir nuevos patrones en el odio que no se habían reflejado previamente en los datos originales. Además, aumentaría la cantidad de datos disponibles para los modelos, haciendo que sean más robustos y generalizables.

## 12 Conclusiones

Este proyecto tenía como objetivo estudiar el odio para encontrar cómo se comporta y la creación de un modelo para poder detectar estos comportamientos tóxicos. Se puede decir que este proyecto ha dado sus frutos y el objetivo ha sido conseguido con éxito.

Se ha llegado a la conclusión de que, en el contexto de las elecciones europeas del año 2024, el odio se usa claramente como instrumento con intención política, y se ha podido identificar mucho más odio hacia los partidos políticamente situados hacia la izquierda española, lo que puede deberse a que forman parte del gobierno, atrayendo más crítica que otros partidos que actualmente no tienen tanta relevancia en la toma de decisiones. Una posible línea de futuro de investigación podría centrarse en estudiar si este odio depende de la orientación del partido gobernante o si es constante e independiente de la situación política.

También se ha podido ver que las plataformas tanto de Youtube como de Twitter tienen una moderación suficientemente estricta como para moldear en cierto modo el comportamiento de los usuarios, es decir, se ha confirmado que los comportamientos tóxicos muestran un comportamiento de sistema adversarial adaptativo al sistema de moderación, ya que se adapta a las políticas de comportamiento de la plataforma.

Esta hipótesis se ha podido confirmar al investigar dos hechos. Al analizar la cantidad de palabras y emojis por plataforma se ha podido ver que hay diferencia en el uso de ellas, es decir, que los usuarios se comportan de maneras distintas en Youtube y en Twitter. Esto confirma que el sistema de moderación de cada plataforma elimina ciertos comentarios de los usuarios y deja solamente los que cumplen con las políticas de comportamiento. El segundo hecho que ha confirmado la hipótesis ha sido que, si se clasifican los datos en función de la plataforma, se puede distinguir una diferencia de comportamiento entre los distintos datos, demostrando que sí son datos distintos y que los datos de las plataformas de Youtube y Twitter no se pueden mezclar para crear un dataset más amplio.

Puesto que el comportamiento de los emojis y de los textos depende de la plataforma, se pensó que sería interesante probar modelos que tengan en cuenta tanto los emojis como los textos de manera independiente, para así poder hacer una clasificación con más contexto.

A continuación, se ha revisado el procesamiento de los datos usados y cómo ciertos elementos de los textos pueden añadir ruido al no aportar información útil. Después, se ha visto tanto de manera teórica como con ejemplos prácticos el funcionamiento de los encodings y embeddings. Concretamente: One-hot encoding, TF-IDF, Word2Vec, FastText y BERT.

Después de ver los distintos tipos de embeddings se han estudiado tres modelos de *Machine Learning* utilizando los distintos tipos de embeddings. Los modelos de *Machine Learning* usados han sido SVM, Random Forest y XGBoost. Tras comparar los resultados de cada modelo con cada embedding se ha llegado a la conclusión que el modelo de Random

Forest utilizando embeddings de BETO ha conseguido una f1 de 0.7448, una accuracy de 0.7353 y un valor de curva ROC-AUC de 0.7349. Estos resultados se deben a que los embeddings de BETO son mucho mejores representando las palabras, teniendo en cuenta el contexto y su significado semántico.

Después de ver los modelos de *Machine Learning* se tomó la decisión de continuar con modelos de *Deep Learning* ya que, al ser más complejos, podrían encontrar mejor las relaciones semánticas para hacer la clasificación de textos. Primero se empezó utilizando modelos que utilizaban los embeddings de BERT y BETO (BERT en español) para hacer la clasificación con un *Multi Layer Perceptron* (MLP), y se encontró que, como era de esperar, el hecho de que el modelo esté especializado para entender aumenta el rendimiento de los modelos.

Al continuar el proyecto surgió la necesidad de crear un modelo que tuviera en cuenta el texto y los emojis por separado. Para el procesado de texto se decidió utilizar Redes Neuronales Recurrentes (RNN) que utilizan los embeddings de un modelo BETO, que usa *fine tuning* durante el entrenamiento, y los emojis se codificaron utilizando una fórmula para conseguir una puntuación de sentimiento del comentario o se codificaron usando One-hot encoding. Se vio que los resultados de los modelos no podían superar el valor de f1 de 0.8. dando como mejor resultado el que combina una MLP con una RNN y que asigna una puntuación a los emojis, ya que consigue los mejores resultados con un coste menor.

Al investigar qué datos estaban siendo mal clasificados, se encontró que el etiquetaje de los datos no era consistente. Esto se debe a que el proceso de etiquetaje ha sido realizado con varias personas, provocando discrepancias entre lo que se considera odio o no. Para solucionar este problema se decidió utilizar modelos de lenguaje (LLM) para clasificar los textos y así luego hacer una prueba de *Kendall* y comprobar la consistencia de los etiquetadores. Se utilizaron modelos especializados en la detección de odio y modelos generales. Se concluyó que los modelos especializados en la detección de odio consiguen un rendimiento prácticamente igual al de los etiquetadores humanos. El modelo *mrm8488/bert-tiny-finetuned-sms-spam-detection* consiguió superar los resultados de las personas, consiguiendo aumentar los resultados del modelo que usa MLP con RNN y la puntuación de los emojis de una f1 de 0.7912 a 0.8574. Confirmando la importancia de tener un etiquetaje consistente en los datos para poder conseguir buenas predicciones de los modelos.

En conclusión, este trabajo ha demostrado que el comportamiento en las distintas redes sociales obliga a crear un modelo especializado para cada contexto y ha demostrado la importancia de hacer los sistemas de etiquetaje lo más consistentes posibles ya que, de lo contrario, el resultado es un etiquetaje no muy consistente y que puede ser superado por un modelo de lenguaje. El mejor sistema de detección del odio para las elecciones europeas del 2024 que se ha podido encontrar en este proyecto es el que consta de un etiquetaje que utiliza LLM y cuya arquitectura procesa los textos y los emojis por separado para obtener más contexto, siendo el mejor método el que procesa los textos con BETO y una RNN y luego se usa una capa MLP para clasificar los datos con la puntuación de los emojis y los resultados del contexto semántico de la RNN.

## 13 Bibliografía

- Abushnama. (04 de 03 de 2023). *Understanding Gated Recurrent Unit (GRU) in Deep Learning.* Obtenido de Medium: <https://medium.com/@anishnama20/understanding-gated-recurrent-unit-gru-in-deep-learning-2e54923f3e2>
- Alathur, N. C. (2018). Hate speech review in the context of online social networks. 108-118.
- Aymé Arango, J. P. (2019). *Hate Speech Detection is Not as Easy as You May Think.* Paris: ACM Digital Library.
- Baheti, P. (27 de 3 de 2021). *7labs.* Obtenido de 7labs: <https://www.v7labs.com/blog/neural-networks-activation-functions>
- Bigdata, P. (12 de 06 de 2025). *HuggingFace.* Obtenido de HuggingFace: <https://huggingface.co/piuba-bigdata/beto-contextualized-hate-speech>
- Breiman, L. (1996). Bagging Predictors. *Kluwer Academic Publishers*, 1-18.
- Breiman, L. (2001). Random Forests. *Springer Nature Link*, 5-32.
- Burkov, A. (2025). *The hundred-page language models book hands-on with PyTorch.*
- Cañete, J. a.-H. (22 de 06 de 2025). *HuggingFace.* Obtenido de dccuchile/bert-base-spanish-wwm-uncased: <https://huggingface.co/dccuchile/bert-base-spanish-wwm-uncased>
- Chaudhay, A. (21 de 06 de 2020). *Amit Chaudhay.* Obtenido de Amitness: <https://amitness.com/posts/fasttext-embeddings>
- Copeland, B. (19 de 04 de 2025). *Britanica.* Obtenido de history of artificial intelligence (AI): <https://www.britannica.com/science/history-of-artificial-intelligence>
- Dancker, J. (26 de 12 de 2022). *A brief introduction to Recurrent Neural Networks.* Obtenido de Medium: <https://medium.com/data-science/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4>
- DataTab. (12 de 06 de 2025). *What is the p-value?* Obtenido de DataTab: <https://datatab.net/tutorial/p-value>
- deepseek-ai. (20 de 06 de 2025). *HuggingFace.* Obtenido de HuggingFace: <https://huggingface.co/deepseek-ai/DeepSeek-R1-0528-Qwen3-8B>
- E.Schapire, Y. F. (1999). A short introducction to Boosting. *Journal of Japanise Society for Artificial Intelligence*, 1-14.
- Facebook. (11 de 06 de 2025). *fastText.* Obtenido de fastText: <https://fasttext.cc/>
- Facebook. (11 de 06 de 2025). *GitHub.* Obtenido de GitHub: <https://github.com/facebookresearch/fastText>
- Gomez, P. X. (2024). *Detección de discurso de odio dirigido a las mujeres en la plataforma Tik Tok.* Barcelona: La Salle-Universitat Ramon Llull.

- Google. (12 de 06 de 2025). *Clasificación: Exactitud, recuperación, precisión y métricas relacionadas* . Obtenido de Google Developers: <https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall?hl=es-419>
- Google. (12 de 06 de 2025). *Redes neuronales*. Obtenido de Google developers: <https://developers.google.com/machine-learning/crash-course/neural-networks?hl=es-419>
- Google Cloud. (17 de 03 de 2025). *Youtube*. Obtenido de [https://developers.google.com/youtube/v3/determine\\_quota\\_cost?hl=es-419](https://developers.google.com/youtube/v3/determine_quota_cost?hl=es-419)
- Google developers . (09 de 06 de 2025). *Datos categóricos: Vocabulario y codificación one-hot*. Obtenido de Google developers machine learning: <https://developers.google.com/machine-learning/crash-course/categorical-data/one-hot-encoding?hl=es-419>
- Google developers. (09 de 06 de 2025). *Google developers*. Obtenido de Prepare sus datos: <https://developers.google.com/machine-learning/guides/text-classification/step-3?hl=es-419>
- Guestrin, T. C. (2016). XGBoost: A scalable Tree Boosting System. *arXiv*, 1-13.
- Guterres, U. N.-G. (2019). United nations strategy and plan of action on hate speech. 1-5.
- Jacob Devlin, M.-W. C. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv Cornell Universiry* , 1-16.
- Jordan, M. I. (1986). Serial order: A parallel distributed processing approach. *ICS*, 1-46.
- José Cañete, G. C. (2020). SPANISH PRE-TRAINED BERT MODEL AND EVALUATION DATA. *ArXiv*, 1-9.
- Juan Carlos Pereira-Kohatsu, L. Q.-S.-C. (2019). *Detecting and Monitoring Hate Speech in Twitter*. Sensors.
- Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv*, 1-9.
- Karabiber, F. (09 de 06 de 2025). *LearnDatascience*. Obtenido de <https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/>
- Kendall, M. G. (1938). A New Measure of Rank Correlation. *jstor*, 81-93.
- Lemos, A. R. (14 de 04 de 2022). *Towards Data Science*. Obtenido de <https://towardsdatascience.com/cross-validation-705644663568/>
- Louis, A. (06 de 07 de 2020). *Medium*. Obtenido de Medium: <https://medium.com/@antoine.louis/a-brief-history-of-natural-language-processing-part-2-f5e575e8e37>
- malik, s. (20 de 07 de 2024). *Medium*. Obtenido de Medium: <https://medium.com/@satyamlkinf/the-nlp-landscape-from-1960-to-2024-a-journey-through-time-7f52c67872fb>

- ManyTools. (17 de 06 de 2025). *ManyTools*. Obtenido de ManyTools: <https://manytools.org/hacker-tools/ascii-banner/>
- María Antonia Paz, J. M.-D.-D. (2020). Hate Speech: A Systematized Review. *Sage Journals*.
- Martin, R. C. (2000). Desing principles and design patterns. *ObjectMentor*, 1-34.
- Md Mustafizur Rahman, D. B. (2021). An Information Retrieval Approach to Building. *Cornell University*.
- Md Saroor Jahana, M. O. (2017). *A Comprehensive Study on NLP Data Augmentation for Hate Speech Detection: Legacy Methods, BERT and LLMs*. Oulu, Finland.
- Meta. (12 de 06 de 2025). *HuggingFace*. Obtenido de HuggingFace: <https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct>
- mrm8488. (19 de 06 de 2025). *HuggingFace*. Obtenido de HuggingFace: <https://huggingface.co/mrm8488/bert-tiny-finetuned-sms-spam-detection>
- Mudit Chaudhary, C. S. (2021). *Countering Online Hate Speech: An NLP Perspective*.
- N. V. Sahana, P. R. (2020). *Automatic Hate Speech Detection using Ensemble Method and Natural Language Processing Techniques*. Bangalore, India.
- natural, A. s. (2021). *Md Saroor Jahan and Mourad Oussalah*. ElServier.
- omkar-foss. (2023). *github.com*. Obtenido de GitHub: <https://github.com/omkar-foss/emosent-py/blob/master/README.md>
- Paula Fortuna, M. D. (s.f.). *Directions for NLP Practices Applied to Online Hate Speech Detectio*.
- PcPartPicker. (21 de 06 de 2025). *PcPartPicker*. Obtenido de <https://pcpartpicker.com/>
- Petra Kralj Novak, J. S. (2015). Sentiment of Emojis. *PLOS One*, 1-22.
- Piotr Bojanowski, E. G. (2026). Enriching Word Vectors with Subword Information. *arXiv*, 1-12.
- pysentimiento. (19 de 06 de 2025). *HuggingFace*. Obtenido de HuggingFace: <https://huggingface.co/pysentimiento/robertuito-hate-speech>
- Ramis, O. (2022). *Estudi dels tweets d'odi dirigits a periodistes*. Barcelona: La Salle-Universitat Ramon Llull.
- Rosenblatt, F. (1958). The perceptron a probabilistic model for information storage and organization in the brain. *Psychological Review*, 1-23.
- Sánchez, J. d. (12 de 06 de 2025). *HuggingFace*. Obtenido de HuggingFace: <https://huggingface.co/delarosajav95/HateSpeech-BETO-cased-v2>
- ScikitLearn. (10 de 06 de 2025). *ScikitLearn*. Obtenido de ScikitLearn: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
- Sellars, A. (2016). Defining Hate Speech. 33.
- Sellars, A. F. (s.f.). Defining hate speech. *Berkman Klein Center - Harvard university*.

- Servera, L. (2024). *Creació i anàlisi d'una base de dades per a l'estudi de llenguatge d'odi contra periodistes en base a tècniques de NLP*. Barcelona: La Salle-Universitat Ramon Llull.
- Siddiqi, M. a. (2017). *Automatic Hate Speech Detection: A Literature Review*. India.
- Singh, V. (31 de 08 de 2023). *Shiksha online*. Obtenido de Difference Between Bagging and Boosting: <https://www.shiksha.com/online-courses/articles/bagging-and-boosting/>
- The Bridge. (12 de 12 de 2024). *The Bridge*. Obtenido de Salario medio de un Data Scientist en España en 2025: <https://thebridge.tech/blog/salario-data-scientist-espana/>
- till-m. (10 de 06 de 2025). *GitHub*. Obtenido de GitHub BayesianOptimization: <https://github.com/bayesian-optimization/BayesianOptimization>
- Tomas Mikolov, K. C. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv*, 1-12. Obtenido de <https://arxiv.org/abs/1301.3781>
- Vapnik, C. C. (1995). Support-vector networks. *Springer Nature Link*, 273-297.
- Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 36-45.
- Wikipedia. (04 de 06 de 2025). *Decision tree learning*. Obtenido de Wikipedia: [https://en.wikipedia.org/wiki/Decision\\_tree\\_learning](https://en.wikipedia.org/wiki/Decision_tree_learning)
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *ArXiv*, 1-13.
- Youtube. (17 de 03 de 2025). *Youtube Data API*. Obtenido de <https://developers.google.com/youtube/v3/docs?hl=es-419>

# 14 Tablas

## 14.1 Tabla de ilustraciones

Ilustración 1: línea temporal del progreso del NLP .....	15
Ilustración 2: métricas modelos entrenados con y sin palabras de odio .....	20
Ilustración 3: Proceso de filtraje de mensajes de odio.....	21
Ilustración 4: Arquitectura implementada en estudio UAM .....	22
Ilustración 5: gráfica de odio separado por zona geográfica .....	26
Ilustración 6: gráfica de odio por partido político .....	27
Ilustración 7: gráfica de odio por políticos españoles .....	28
Ilustración 8: gráfica de odio por políticos europeos.....	29
Ilustración 9: gráfica de odio por ideología .....	30
Ilustración 10: gráfica de odio por comunidad autónoma .....	31
Ilustración 11: wordcloud de todas las palabras.....	33
Ilustración 12: wordcloud de todas las palabras separadas por plataforma.....	34
Ilustración 13: gráfica de barras de palabras 1-15 separadas por plataforma .....	35
Ilustración 14: gráfica de barras de palabras 16-30 separadas por plataforma.....	35
Ilustración 15: wordcloud de comentarios con odio por plataforma .....	36
Ilustración 16: gráfica de barras de odio de palabras 1-15 separadas por plataforma .....	36
Ilustración 17: gráfica de barras de odio de palabras 16-30 separadas por plataforma....	37
Ilustración 18: wordcloud de comentarios sin odio por plataforma .....	38
Ilustración 19: gráfica de barras sin odio de palabras 1-15 separadas por plataforma .....	38
Ilustración 20: gráfica de barras sin odio de palabras 16-30 separadas por plataforma ...	39
Ilustración 21: emojicloud de todos los emojis.....	40
Ilustración 22: emojicloud con los emojis que aparecen más de 500 veces .....	41
Ilustración 23: emojicloud con los emojis que aparecen más de 200 veces .....	41
Ilustración 24: suma de emojis .....	42
Ilustración 25: emojicloud con los emojis que aparecen más de 200 veces sin etnia .....	42
Ilustración 26: emojicloud de emojis positivos .....	43
Ilustración 27: emojicloud de emojis negativos .....	43
Ilustración 28: emojicloud de emojis neutros .....	43
Ilustración 29: gráfica de barras de emojis 1-15 separadas por plataforma .....	44
Ilustración 30: gráfica de barras de emojis 16-30 separadas por plataforma .....	44
Ilustración 31: treemap de los emojis más usados .....	45
Ilustración 32: gráfica de barras de emojis con odio 1-15 separadas por plataforma .....	46
Ilustración 33: gráfica de barras de emojis con odio 16-30 separadas por plataforma .....	46
Ilustración 34: gráfica de barras de emojis sin odio 1-15 separadas por plataforma .....	48
Ilustración 35: gráfica de barras de emojis sin odio 16-30 separadas por plataforma .....	48
Ilustración 36: representación de las palabras en word2vec .....	55
Ilustración 37: ejemplo de división de palabras en trigramas .....	56
Ilustración 38: Estructura BERT .....	57
Ilustración 39: ejemplo matriz de confusión.....	59
Ilustración 40: proceso de optimización bayesiana .....	61
Ilustración 41: ilustración funcionamiento cross validation .....	61

Ilustración 42: representación del funcionamiento de un SVM .....	62
Ilustración 43: ejemplo de árbol de decisión .....	64
Ilustración 44: esquema del funcionamiento de bagging .....	65
Ilustración 45: esquema del funcionamiento de boosting .....	67
Ilustración 46: Imagen comparativa entre neurona biológica y artificial.....	71
Ilustración 47: esquema red neuronal simple (imagen propia) .....	72
Ilustración 48: Estructura MLP .....	73
Ilustración 58: arquitectura de RNN y GRU.....	75
Ilustración 49: Gráfica accuracy y loss MLP.....	76
Ilustración 50: Matriz de confusión MLP .....	77
Ilustración 51: Gráfica accuracy y loss MLP con dropout .....	78
Ilustración 52: Matriz de confusión MLP con dropout.....	79
Ilustración 53: MLP con dropout bajo.....	80
Ilustración 54: Gráfica accuracy y loss MLP reducido con dropout .....	81
Ilustración 55: Matriz de confusión MLP reducido con dropout .....	81
Ilustración 56: Matriz de confusión MLP arquitectura #3 + SVM .....	82
Ilustración 57: Matriz de confusión MLP arquitectura #3 + SVM + GridSearch .....	83
Ilustración 59: matriz de confusión RNN .....	87
Ilustración 60: proceso de entrenamiento de la RNN .....	87
Ilustración 61: pipeline de procesado de datos .....	89
Ilustración 62: diagrama procesado BETO.....	90
Ilustración 63: diagrama entrenamiento RNN con embeddings BETO.....	91
Ilustración 64: diagrama estructura MLP con representación contextual y puntuación ...	91
Ilustración 65: diagrama proceso entrenamiento RNN+MLP .....	91
Ilustración 66: proceso de entrenamiento de la RNN+MLP .....	93
Ilustración 67: matriz de confusión RNN+MLP.....	94
Ilustración 68: BETO con MLP .....	95
Ilustración 69: matriz de confusión de BETO+MLP .....	96
Ilustración 70: clasificación manual en función de la puntuación de los emojis .....	96
Ilustración 71: clasificación con ML en función de la puntuación de los emojis .....	97
Ilustración 72: estructura RNN+MLP+OneHotEmoji .....	98
Ilustración 73: matriz de confusión del modelo de MLP+RNN con one-hot encoding .....	99
Ilustración 74: entrenamiento de RNN+MLP con emojis con one-hot encoding .....	100
Ilustración 75: arquitectura modelo clasificador de plataforma .....	101
Ilustración 76: entrenamiento de clasificador de plataforma .....	101
Ilustración 77: matriz de confusión del clasificador de plataforma .....	102
Ilustración 78: Diagrama pipeline datos y modelos.....	104
Ilustración 79: diagrama de actividad del pipeline de modelos .....	107
Ilustración 80: menú principal del pipeline.....	108
Ilustración 81: finalización del proceso de procesado de datos.....	109
Ilustración 82: resultado del experimento esBERT .....	110
Ilustración 83: resultado del entrenamiento .....	110
Ilustración 84: accuracy y pérdida del estudio esBERT uncased con batch size 16.....	111
Ilustración 85: matriz de confusión del estudio esBERT uncased con batch size 16 .....	111
Ilustración 86: arquitectura del modelo del artículo esBERT .....	114
Ilustración 87: arquitectura del modelo del artículo RoBERTa .....	116
Ilustración 88: diagrama de Gantt .....	124

## 14.2 Tabla de tablas

Tabla 1: ejemplo one-hot encoding .....	53
Tabla 2: resultados SVM con los distintos embeddings .....	63
Tabla 3: resultados DecisionTree con los distintos embeddings .....	66
Tabla 4: resultados RandomForest con los distintos embeddings .....	66
Tabla 5: resultados XGBoost con los distintos embeddings.....	68
Tabla 6: comparación f1 final de modelos ML con distintos embeddings .....	69
Tabla 7: comparación accuracy final de los modelos ML con distintos embeddings.....	69
Tabla 8: comparación precision final modelos ML con distintos embeddings .....	69
Tabla 9: comparación recall final modelos ML con distintos embeddings .....	69
Tabla 10: comparación ROC-AUC final modelos ML con distintos embeddings .....	70
Tabla 11: resultados del modelo rnn en los distintos datasets.....	86
Tabla 12: ejemplo de tokenización de emojis.....	89
Tabla 13: ejemplos de la aplicación de la ecuación 1.....	90
Tabla 14: resultados del modelo rnn+mlp en los distintos datasets .....	92
Tabla 15: clasificación de los modelos en función de la puntuación de los emojis .....	97
Tabla 16: resultados del modelo rnn+mlp+one-hot en los distintos datasets .....	99
Tabla 17: resultados del modelo clasificador de plataforma .....	102
Tabla 18: resultados del modelo clasificador de plataforma con emojis .....	103
Tabla 19: resultados del modelo esBERT en los distintos datasets .....	114
Tabla 20: resultados del modelo RoBERTa en los distintos datasets .....	116
Tabla 21: prompt con los criterios de Blanquerna .....	119
Tabla 22: resultados LLM especializados en detección de odio .....	120
Tabla 23: resultados LLM generativos.....	121
Tabla 24: resultados de RNN+MLP en dataset mrm8488.....	121
Tabla 25: resultados LLM especializados en detección de odio .....	122
Tabla 26: resultados LLM generativos.....	123

## 14.3 Tabla de ecuaciones

Ecuación 1: fórmula TF.....	54
Ecuación 2: fórmula IDF .....	54
Ecuación 3: fórmula TF-IDF .....	54
Ecuación 4: fórmula del accuracy.....	59
Ecuación 5: fórmula del recall .....	59
Ecuación 6: fórmula de la precisión.....	59
Ecuación 7: fórmula de la puntuación de los emojis .....	90

## 15 Apéndice

### 15.1 Tabla de estudio del estado del arte

En las siguientes páginas se puede ver una tabla resumen de los trabajos previos analizados. Para cada trabajo se han analizado los siguientes puntos: nombre del proyecto, autor, origen de los datos, modelos aplicados, resultados obtenidos del modelo y finalmente las conclusiones que se extraen del estudio.

Gracias a este estudio se ha podido encontrar una ruta clara para poder abarcar el reto propuesto.

Nombre	Autor	Origen datos	Modelos	Resultados (accuracy)	Conclusiones proyecto	
<b>TFG</b>  HateSpeech Creació i anàlisi d'una base de dades per a l'estudi de llenguatge d'odi contra periodistes en base a tècniques de NLP	Luis Ángel	X	Pysentimiento  Naive-Bayes  Random Forest  Decision Tree  Logistic Regression	83.74%  77.24%  95.07%  93.16%  <b>95.41%</b>	Se necesitan más posts de odio (el 94% de los datos son de tipo no odio). Todos los modelos clasifican bien cuando no hay odio, pero clasifican muy mal cuando sí que es odio.	
<b>TFM</b>  Detección de discurso de odio dirigido a las mujeres en la plataforma Tik Tok	Paula Ximena	X TikTok	175,399 tuits, 48% de odio y 52% no ofensivos. Word embeddings: Word2Vec y BERT	Logistic Regression  Logistic Regression con L2  KNN  SVM  SVM con RBF  SVM con polinomio Kernel  Naive Bayes  Complement Naive Bayes  XGBoost  XGBoost con ajuste  LSTM	87%  88%  87%  87%  81%  74%  89%  85%  87%  88%  88%	30.5% de datos son odio y 69.5% de datos sin odio.  <b>Naive bayes consigue muy buenos resultados a pesar de su simplicidad</b> , es eficiente al compararlo con los otros modelos.  <b>LSTM consigue resultados especialmente robustos</b>

<b>TFM</b>  Madrid 4M Estudi dels tweets d'odi dirigits a periodistes	Oriol Ramis	Twitter	Clasificación binaria	Naive Bayes	91,31%	<b>El dispositivo móvil parece tener influencia</b> en el odio. El sexo del periodista que publica el post influye ( <b>el odio es un 5,5% superior en mujeres</b> , siendo solo el 42% del total). Se tiene que contemplar que <b>muchas cuentas son de bots</b> (en este caso el 35%).
				Random Rorest	100%	
				XGBoost	94,95%	
				SVM	99,98%	
				W2Vec con Bayes	41,45%	
				W2Vec con Random forest	99,16%	
				W2Vec con XGBoost	98,96%	
				W2Vec con SVM	99,98%	
				CNN	31,21%	
				RNN	36,60%	
<b>Artículo</b>  Detecting and monitoring Hate Speech in Twitter	Juan Carlos Pereira-Kohatsu, Lara Quijano-Sánchez, Federico Liberatore and Miguel Camacho-Collados	Twitter	Threshold de 0.5	Naive Bayes	97,88%	El mejor resultado ha sido con LTSM+MLP con threshold de 0.7 <b>Es importante tener en cuenta los emojis</b> , pueden dar contexto.
				Random Rorest	100%	
				XGBoost	97,82%	
				SVM	100%	
				W2Vec con Bayes	51,14%	
				W2Vec con Random forest	97,88%	
				W2Vec con XGBoost	97,88%	
				W2Vec con SVM	81%	
				CNN	31%	
				RNN	31,60%	
			Threshold de 0.7	LTSM+MLP (Words, emojis, expression tokens, y tf-idf)	63,5%	
				LTSM+MLP (Words, emojis, expression tokens, y tf-idf)	<b>78,4%</b>	
				Ridge R.	79.4%	
				SVM	78,6%	
				...		

<b>Artículo</b> A systematic review of hate speech automatic detection using natural language processing	Md Saroar Jahan and Mourad Oussalah	Twitter	Modelos comentados en el siguiente apartado	Modelos comentados en el siguiente apartado	<b>En la sección 7 se indica otros datasets con Tweets (todos supervisados)</b>
<b>Artículo</b> Hate Speech Detection Using Natural Language Processing Techniques	Shanita Biere	Twitter	CNN	91% Recall: 90% F1-score: 90% Precisión: 91%	<b>Aunque los resultados sean buenos, muchos tweets considerados como HS no se han clasificado bien.</b>
<b>Artículo</b> Automatic Hate Speech Detection: A Literature Review	Mohiyaddeen and Dr. Sifatullah Siddiq	Facebook, MySpace, YouTube, Kaggle and Twitter	TF-IDF + Naive Bayes	93 %	Resaltan la necesidad de tener un dataset grande para poder hacer que los modelos aprendan de verdad.
			TF-IDF + Métodos essemble	83.4 %	
			TF-IDF + Multinomial Logistic Regression	87.68 %	
			TF-IDF + KNN	<b>97.19 %</b>	
			SentiWordNet, VADER	96 %	
			CNN-LSTM, Twitter Embedding	Precision: 84.5%	
<b>Artículo</b>		Twitter	XGBoost Classifier	87 %	

<p>Automatic Hate Speech Detection using Ensemble Method and Natural Language Processing Techniques</p>	<p>N. V. Sahana, Prerana R., Niharika S., Rakshitha S. and Bhanushree K. J.</p>	<p>Weak learners</p>	<table border="1"> <tbody> <tr> <td>Random Forest Classifier</td><td><b>88 %</b></td></tr> <tr> <td>Naive Bayes Classifier</td><td>86 %</td></tr> <tr> <td>Logistic Regression</td><td>86 %</td></tr> <tr> <td>Decision Tree Classifier</td><td>84 %</td></tr> <tr> <td>Support Vector Classifier</td><td><b>88 %</b></td></tr> </tbody> </table>	Random Forest Classifier	<b>88 %</b>	Naive Bayes Classifier	86 %	Logistic Regression	86 %	Decision Tree Classifier	84 %	Support Vector Classifier	<b>88 %</b>	<p>Se ha observado que los métodos ensamble funcionan mejor cuando los textos pueden ser de distintos idiomas. Comenta que se debe tener en cuenta las limitaciones del dataset usado en cuanto a tamaño, diversidad y generalización al usarlo en contextos con múltiples lenguas.</p>
Random Forest Classifier	<b>88 %</b>													
Naive Bayes Classifier	86 %													
Logistic Regression	86 %													
Decision Tree Classifier	84 %													
Support Vector Classifier	<b>88 %</b>													
<p><b>Artículo</b>  Trends in Machine learning on automatic detection of hate speech on social media platforms a systematic review</p>	<p>Twitter</p>		<table border="1"> <tbody> <tr> <td>Probabilistic, ruled-based and spatial based classifiers</td><td><b>98 %</b></td></tr> <tr> <td>LR, NB, and Linear SVM</td><td>95.6 %</td></tr> <tr> <td>RNN</td><td>92.95 %</td></tr> <tr> <td>LR, NB, and Linear SVM</td><td>95.6 %</td></tr> <tr> <td>Logistic regression</td><td>90 % – 97 %</td></tr> </tbody> </table>	Probabilistic, ruled-based and spatial based classifiers	<b>98 %</b>	LR, NB, and Linear SVM	95.6 %	RNN	92.95 %	LR, NB, and Linear SVM	95.6 %	Logistic regression	90 % – 97 %	<p>Recomiendan que se usen algoritmos no supervisados o semi-supervisados para poder crear un DL híbrido que pueda detectar el odio. Este método soluciona el problema con los emojis y abreviaciones no estándar. Comenta mejor no usar met. sup. para evitar etiqu.</p>
Probabilistic, ruled-based and spatial based classifiers	<b>98 %</b>													
LR, NB, and Linear SVM	95.6 %													
RNN	92.95 %													
LR, NB, and Linear SVM	95.6 %													
Logistic regression	90 % – 97 %													

## 15.2 Criterios de clasificación de Blanquerna

El siguiente fragmento de texto ha sido proporcionado por Jaume Suau, representante de Blanquerna en el proyecto Disargue.

### Typology of Online Attacks on Journalists

#### Category 1: Incivism, Verbal Abuse, and Insults

- Definition: Tweets that refer to the journalist using bad words, insults, or derogatory terms. This category includes attacks on professional integrity but excludes claims of violence, personal aggression, or references to personal conditions as a member of a particular social group.

- Examples:

- "Brian, you are a bad journalist always serving those in power."
- "Ignacio Escolar eres un cabrón."
- "Ana Pastor eres una cabrona" (not referring to her gender or sexual identity).

#### Category 2: Online Vitriol, Threats, and Intimidation

- Definition: This category includes tweets with a more aggressive tone that contain threats or mentions of violence, whether implicit or explicit.

- Examples:

- "Brian, if you continue writing like this, someone will teach you a lesson."
- "I hope something bad happens to you for spreading lies."

#### Category 3: Hate Speech

- Definition: Tweets that attack the journalist based on their membership in a particular social group, including gender, race, ideology, LGBTQ+ identity, etc.

- Examples:

- "Ana Pastor eres una puta" (attacking based on gender).
- "You should go back to your country, you don't belong here" (attacking based on race or ethnicity).
- "You gay journalists are ruining everything" (attacking based on sexual orientation).

#### Category 4: Comments about the physical appearance of the journalist

- Definition: Tweets that mention the appearance of the journalist and that are not related to his/her job nor to the original tweet. These tweets are not “hate” or “attacks” as defined in category 1 or 2, but are focused on the physical appearance of the journalist, deviating from his/her job.

- Examples:

- "Ana Pastor eres muy guapa /eres preciosa /eres un bellezón...".
- "Estoy enamorado de ti / me encantaría estar contigo" (does not mention physical appearance directly but still considered in this category).

NOTE: In this category we will find basically i) creepy guys commenting on the physical appearance of female journalists; ii) women that tell the journalist how beautiful she is as a kind of nice or supportive comment. The tool cannot distinguish that but will give us a dataset of the tweets labeled as 1-4 that then we can use to manually detect which case is each tweet.

#### Guidelines for Classification

1. **\*\*Tweets with Multiple Aspects:\*\*** If a tweet includes elements from multiple categories, it should be labeled under the highest category present. For instance, if a tweet contains both incivism (Category 1) and hate speech (Category 3), it should be classified as Category 3. Only exception is Category 4 which is considered as 0. For example, a tweet that states "Eres muy guapa pero tonta" is a 1-1.

2. **\*\*Clear Differentiation:\*\*** Ensure that tweets are categorized based on the specific criteria outlined above. For example, an insult that does not reference gender or other personal conditions should remain in Category 1.

#### Application of the Typology

##### Example 0: No category, classified as No Attack

- Tweet: "Brian, you are a bad journalist"
- Reasoning: This is just an opinion, journalists can be criticized, as any other person or profession. Although this might be disrespectful cannot be considered an online attack.

##### Example 1: Category 1

- Tweet: "Brian, you are a bad journalist serving always those in power."
- Reasoning: Insulting professional integrity without violence or personal condition references.

##### Example 2: Category 2

- Tweet: "Brian, if you continue writing like this, someone will teach you a lesson."
- Reasoning: Contains a threat implying potential violence.

##### Example 3: Category 3

- Tweet: "Ana Pastor eres una puta."
- Reasoning: Attacks the journalist based on gender.

### 15.3 Criterios de clasificación de Pere Vidal

Pere Vidal es un estudiante del máster de MUDS online que está estudiando la detección de odio orientado hacia las mujeres durante el 8M en la plataforma de TikTok. Para poder hacer la clasificación de los comentarios ha usado LLM para luego hacer la comparativa de clasificadores. El *prompt* que ha utilizado es el siguiente:

sistema = f""""

En base a la frase proporcionada por el usuario, determina si esta frase se clasificaría como discurso de odio dirigido a las mujeres según la siguiente definición:

El discurso de odio es toda comunicación escrita que tiene como propósito atacar, discriminar, minimizar, incentivar a la violencia física o la hostilidad. Utiliza palabras peyorativas dirigidas a una persona o unos grupos que defienden públicamente los derechos de las mujeres.

A continuación, se describen los tipos de discurso de odio más comunes:

1. Discriminación: Comentarios que sugieren que el respeto no es universal para todas las mujeres, sino que depende de la aprobación o sintonía con ciertos valores o ideas.
2. Insultos Misóginos: Comentarios que utilizan términos despectivos basados en el género para denigrar a una mujer.
3. Amenazas de Violencia Sexual: Mensajes que incluyen amenazas explícitas de violación u otras formas de agresión sexual.
4. Descalificaciones por Apariencia Física: Críticas ofensivas sobre el cuerpo o la apariencia de una mujer.
5. Negación de Capacidades Intelectuales: Afirmaciones que cuestionan la inteligencia o competencia de una mujer debido a su género.
6. Comentarios Sexistas sobre Roles de Género: Expresiones que relegan a las mujeres a roles tradicionales.
7. Descalificación de Logros Profesionales: Atribuir el éxito de una mujer a favores sexuales o minimizar sus logros.
8. Humillación Pública por Decisiones Personales: Criticar decisiones personales, como la maternidad o la elección de no tener hijos.
9. Difusión de Estereotipos de Género: Comentarios que perpetúan estereotipos negativos.
10. Incitación al Odio Colectivo: Mensajes que fomentan el odio hacia las mujeres en general.
11. Deshumanización: Comparar a las mujeres con animales u objetos para degradarlas.

12. Desacreditación Profesional: Cuestionar la legitimidad de una mujer en su ámbito laboral o académico basándose únicamente en su género, insinuando que no merece su posición.

13. Negación de Experiencias de Discriminación: Minimizar o ridiculizar las experiencias de una mujer relacionadas con el machismo o la violencia de género, sugiriendo que exagera o miente.

14. Ridiculización de Movimientos Feministas: Burlarse o desprestigar iniciativas que buscan la igualdad de género, etiquetándolas de manera peyorativa.

Responde solamente "Sí" o "No" junto con cuál de los 14 tipos corresponde

Ejemplo de formato de respuesta admitida para "No": "No"

Ejemplo de formato de respuesta admitida para "Sí": "Sí. 1. Discriminación"

.....

## 15.4 Artículo CCIA edición 27<sup>a</sup>

Durante la realización de este proyecto se ha redactado un artículo para participar en la edición 27 del **Congreso Internacional Asociación Catalana de Inteligencia Artificial (CCIA)** del año 2025.

En el artículo se investigan la aportación de los emojis en el proceso de clasificación de texto con redes neuronales recurrentes (RNN). Para ello se analiza el comportamiento de los siguientes modelos: RNN (usando embeddings de BATO), RNN haciendo *fine tuning* a BATO, el modelo RNN + MLP (con puntuación para codificar los emojis) y RNN + MLP (con *one hot encoding* para codificar los emojis).

Para determinar si las diferencias entre los resultados obtenidos por los distintos modelos son estadísticamente significativas se utilizó el test ANOVA. Posteriormente, se utilizó el test Tukey HSD para identificar que modelos difieren entre sí de forma significativa.

Los resultados de estos análisis indican que existen diferencias estadísticamente significativas entre los modelos, con una delta de 0.0008. El modelo que obtuvo mejor rendimiento fue *RNN+MLP (scores)*.

A continuación, se puede ver dicho artículo:

May 2025

# Hate Speech and Emojis: the journalist case

David LARROSA-CAMPS<sup>a</sup>, Jaume SUAU<sup>b</sup> and Xavier VILASÍS-CARDONA<sup>b</sup>

<sup>a</sup>Smart Society Research Group, La Salle-Universitat Ramon Llull

<sup>b</sup>Digilab, Blanquerna-Universitat Ramon Llull

**Abstract.** Work in progress towards a simple and efficient use of emojis for hate speech detection is reported. The dataset used has been generated in the framework of studying hate speech against journalists. The results of the method are promising, yet not fully conclusive.

**Keywords.** Natural Language Processing, Hate Speech, Emojis

## 1. Introduction

Hate speech, which is strongly related with the spread of disinformation [1], has become one of the biggest challenges that democratic societies are facing today. The relation between disinformation and hate speech in journalism is twofold. First, attacks on and threats to journalists and the media constrain freedom of expression and dissemination of certain information, and generate fear and self-censorship, undermining the media's role in democratic societies [2]. Second, disinformation campaigns often exploit hate narratives to amplify polarization, delegitimize the press, and erode trust in democratic institutions.

For reasons such as the ones described above, hate speech detection has become a pivotal challenge in natural language processing (NLP). Recent advancements leverage transformer-based models, multimodal analysis, and fairness-aware techniques to improve detection accuracy while addressing biases [3,4]. Despite progress, challenges such as contextual ambiguity, cross-lingual generalization, and ethical trade-offs persist.

State-of-the-art hate speech detection systems predominantly rely on transformer architectures such as BERT [5] and RoBERTa [6], which excel at capturing implicit hate speech and sarcasm through contextual embeddings. Fine-tuning these models on domain-specific datasets (e.g., Gab, Twitter) has proven effective, though performance drops in cross-platform scenarios [7]. Spanish hate speech detection presents unique challenges due to linguistic diversity (e.g., regional dialects, Spanglish) and cultural context. Models like BETO [8] (a Spanish BERT variant) and multilingual approaches (e.g., XLM-T [9]) have shown promise, but performance varies across regions (e.g., Mexico vs. Spain).

In our analysis of data from the X platform in search for hate speech against journalists in Spain, we have remarked the heavy use of emojis. Emojis, faithful to

*May 2025*

the name, represent the emotions of the writer and are a valuable tool to identify hate speech. However, they are not part of the grammar of natural language, with which recurrent or transformer models have been trained. Because of this, these models may be confused by their presence. In this note, we investigate a strategy to use them in the detection task, combined with text analysis.

## 2. Data Collection

The data were collected from the social media platform X (formerly Twitter), chosen due to its high levels of toxicity and the presence of all selected journalists. The data collection process took place during the 2024 European elections and the days immediately following. A curated list of journalists to monitor was created, ensuring balance in terms of sector, gender, and other relevant criteria. Tweets and replies were automatically downloaded using Python scripts via the official X API. The endpoints used included user lookup, user tweets, and user mentions.

In total, 41,791 tweets were manually labeled, with only 4.6% (1,947 tweets) identified as containing hate speech, resulting in a highly imbalanced dataset. The labeling was carried out by four annotators, each of whom labeled approximately 10,447 tweets. To address the class imbalance problem, the majority class was undersampled to match the minority class. This resulted in a final dataset composed of 3,894 tweets, with an equal distribution of 1,947 hateful tweets (50%) and 1,947 non-hateful tweets (50%). Among the non-hateful tweets, only 256 (6.57%) contained emojis, while 1,691 (43.43%) did not. Similarly, among the hateful tweets, 259 (6.65%) included emojis, while 1,688 (43.35%) did not. Descriptive analysis revealed a higher incidence of hate speech directed toward women and sports journalists.

## 3. Dealing with Emojis

When processing the data, a major challenge was the dual nature of tweets, which often contain both textual content and emojis. While most Natural Language Processing (NLP) techniques are well-suited for textual data, they are not inherently designed to interpret emojis, which convey affective and contextual nuances that can significantly alter the meaning of a message. Ignoring emojis can therefore lead to a substantial loss of sentiment-related information.

To address this issue, the preprocessing pipeline was designed to treat text and emojis as two separate input modalities. The text was cleaned by removing user mentions, retweet markers, punctuation, numbers, and any emojis. This step ensured that only clean textual content was passed to the language processing models. For the emoji data, the process involved extracting all emojis from the original tweet and assigning each one a sentiment score. To obtain these scores, we used the `emosent` Python library<sup>1</sup>, which is based on the sentiment values published in the study "Sentiment of Emojis" [10]. This study provides manually annotated sentiment scores—expressed as numerical continuous values represent-

---

<sup>1</sup><https://github.com/omkar-foss/emosent-py/blob/master/README.md>

ing positive, neutral, or negative sentiment—for a wide range of emojis, based on human perception. However, since the original dataset is from 2015 and many new emojis have been introduced since then, we manually annotated the missing emojis to ensure completeness and consistency in the scoring process.

After assigning sentiment values to each emoji, we computed an aggregated sentiment score for the emoji set in each tweet using a frequency-weighted formula,  $Emoji\ score = \sum_{n=0}^N f(n) \cdot s(n)$ , where  $f(n)$  is the frequency of the  $n$ -th emoji and  $s(n)$  is its sentiment score. This formula takes into account all the emojis present in the tweet, giving more weight to the most frequently used ones, thus emphasizing the predominant sentiment. Finally, the resulting score was normalized to fall within a predefined range, ensuring consistency across tweets regardless of the total number of emojis used.

#### 4. Experiments

Experiments were conducted by splitting the dataset into three blocks: one for training, one for testing, and one for final evaluation, with proportions of 2726, 584, and 584 samples respectively.

We present four different approaches to emoji treatment, all based on a Recurrent Neural Network (RNN). To validate the hypothesis that emojis can improve hate speech classification, the following models were designed: (1) an RNN with BATO embeddings, without emoji information; (2) an RNN with BATO fine-tuned, also without emoji information; (3) an RNN with BATO fine-tuned, incorporating emojis as sentiment scores; and (4) an RNN with BATO fine-tuned, incorporating emojis as one-hot encoded vectors. The models were designed to assess whether incorporating emojis contributes to the classification task. Each model incrementally improves the representation of textual and emoji content by exploring increasingly complex ways of encoding emoji semantics.

	RNN (embeddings)	RNN (fine-tuning BETO)
<b>F1 score</b>	$0.73 \pm 0.17 \times 10^{-3}$	$0.76 \pm 0.12 \times 10^{-3}$
<b>Accuracy</b>	$0.73 \pm 0.17 \times 10^{-3}$	$0.76 \pm 0.12 \times 10^{-3}$
<b>Precision</b>	$0.74 \pm 0.19 \times 10^{-3}$	$0.77 \pm 0.21 \times 10^{-3}$
<b>Recall</b>	$0.74 \pm 0.26 \times 10^{-3}$	$0.75 \pm 0.44 \times 10^{-3}$
<b>ROC-AUC</b>	$0.73 \pm 0.17 \times 10^{-3}$	$0.76 \pm 0.12 \times 10^{-3}$

**Table 1.** Performance comparison: RNN with embeddings vs. fine-tuned BETO.

	RNN + MLP (score)	RNN + MLP (one-hot)
<b>F1 score</b>	$0.76 \pm 0.40 \times 10^{-3}$	$0.76 \pm 0.22 \times 10^{-3}$
<b>Accuracy</b>	$0.76 \pm 0.39 \times 10^{-3}$	$0.76 \pm 0.22 \times 10^{-3}$
<b>Precision</b>	$0.77 \pm 1.06 \times 10^{-3}$	$0.77 \pm 0.77 \times 10^{-3}$
<b>Recall</b>	$0.78 \pm 5.06 \times 10^{-3}$	$0.77 \pm 1.42 \times 10^{-3}$
<b>ROC-AUC</b>	$0.76 \pm 0.37 \times 10^{-3}$	$0.77 \pm 0.22 \times 10^{-3}$

**Table 2.** Performance comparison: RNN + MLP with emoji score vs. one-hot encoding. (2)

May 2025

## 5. Discussion and Conclusion

Treating separately the emojis seems to reinforce the performance of the detection system. However, we cannot consider these results conclusive. The amount of data is scarce, due to the imbalance and may limit the capability of the RNNs. Further data collection and labeling is under way to confirm the analysis. ANOVA and Tukey HSD tests confirmed statistically significant differences between all models ( $p < 0.001$ ), with RNN + MLP (emoji score) outperforming the rest. This model also showed the highest recall, though without introducing bias, as its F1 score remained balanced and differences were not statistically significant.

## 6. Acknowledgments

Authors want to thank Alvaro García-Piquer and Luis Servera for the data collection under the EU project Media Councils in the Digital Age. This work is supported by the project *Disargue*, reference TED2021-130810B-C22, from the Spanish MICINN.

## References

- [1] Anat Ben-David and Ariadna Matamoros Fernández. Hate speech and covert discrimination on social media: Monitoring the facebook pages of extreme-right political parties in spain. *International Journal of Communication*, 10(0), 2016.
- [2] M. Clark and A. Grech. *Journalists Under Pressure: Unwarranted Interference, Fear and Self-censorship in Europe*. Council of Europe, 2017.
- [3] M. Sap, D. Card, S. Gabriel, Y. Choi, and N. A. Smith. The risk of racial bias in hate speech detection. *Proc. ACL*, pages 1668–1678, 2019.
- [4] B. Mathew, P. Saha, S. M. Yimam, C. Biemann, P. Goyal, and A. Mukherjee. Hatexplain: A benchmark dataset for explainable hate speech detection. *Proc. AAAI*, 35:14867–14875, 2021.
- [5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proc. NAACL*, 1:4171–4186, 2019.
- [6] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- [7] C. J. Kennedy, G. Bacon, A. Sahn, and C. von Vacano. Constructing interval variables via faceted rasch measurement and multitask deep learning: A hate speech application. *PLoS ONE*, 15(12):e0243700, 2020.
- [8] J. Cañete, G. Chaperon, R. Fuentes, J.-H. Ho, H. Kang, and J. Pérez. Spanish pre-trained bert model and evaluation data. *Proc. PML4DC at ICLR*, 2020.
- [9] F. Barbieri, L. Espinosa-Anke, and J. Camacho-Collados. Xlm-t: Multilingual language models in twitter for sentiment analysis and beyond. *Proc. LREC*, pages 258–266, 2022.
- [10] Borut Sluban Igor Mozetič Petra Kralj Novak, Jasmina Smailović. Sentiment of emojis. *PLoS ONE* 10(12): e0144296, 2015.