# Centro Universitario de la Defensa
# en la Escuela Naval Militar

**TRABAJO FIN DE GRADO**

*Detección automática de discurso de odio en redes sociales*

**Grado en Ingeniería Mecánica**


**ALUMNO:** Sheila María Pachón de la Torre

**DIRECTOR:** Milagros Fernández Gavilanes


**CURSO ACADÉMICO:** 2020-2021

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

**TRABAJO FIN DE GRADO**

# *Detección automática de discurso de odio en redes sociales*

**Grado en Ingeniería Mecánica**

Intensificación en Tecnología Naval

Cuerpo General

Universidade Vigo

# RESUMEN

Se presenta un enfoque para la creación de un sistema de detección del discurso de odio en las redes sociales, mediante la aplicación de técnicas de procesamiento del lenguaje natural que servirán para entrenar diversos sistemas supervisados. Con este fin, se realiza así la extracción de numerosas características superficiales, incluyendo estas la polaridad de los emojis y conteo de menciones y hashtags; léxicas, con la aportación de TF-IDF y CounterVectorizer; y, además, la implementación de análisis de sentimiento y de sarcasmo. Se experimenta con combinaciones de características para la búsqueda de aquellas que aporten mayor cantidad de información al sistema con la consiguiente mejora en los resultados. Se realiza así en este TFG pruebas muy variadas teniendo en cuenta el abanico de características consideradas y los varios modelos de aprendizaje existentes. Todas estas pruebas se llevaron a cabo sobre un conjunto de tweets de temática yihadista en un entorno multilingüe, centrándose el trabajo en aquellos escritos en inglés y árabe. Finalmente, se realizó una evaluación exhaustiva utilizando características de un modo individual, pero también grupal, sobre los distintos modelos, concluyendo cuáles de ellos son los que proporcionan un mayor rendimiento en este caso de estudio.

# PALABRAS CLAVE

Machine Learning, TF-IDF, SVM, MLP, Discurso de odio

# AGRADECIMIENTOS

A Dña. Milagros Fernández Gavilanes por su orientación, apoyo e ilusión en el desarrollo del presente TFG, así como al pequeño Rufián por su curiosidad en las reuniones del despacho virtual.
A mis padres, hermana y mi Tito Jorge, por ser unos de los pilares fundamentales de mi vida y ayudarme en los momentos de agobio.
A las niñas, las damas de la 421 por hacer que el paso por la escuela sea una de las mejores épocas en mi vida.

# CONTENIDO

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1 INTRODUCCIÓN Y OBJETIVOS

## 1.1 Motivación

Hoy en día resulta tremendamente sencillo publicar casi cualquier cosa, dada la cantidad de contenido no regulado. También resulta fácil presentar la opinión como un "hecho" y hacer declaraciones falsas sin ser cuestionado. En este sentido, en los últimos años se ha observado una proliferación de discurso de odio en Internet, en particular en las redes sociales, gracias a su avance exponencial.

Sin irnos más lejos, en el mes de octubre del pasado año en Francia, un profesor fue degollado por enseñar unas caricaturas de Mahoma, clasificándose este como un asesinato con relación terrorista. Según citan fuentes oficiales, el autor del mismo publicaba en su cuenta de Twitter una imagen con la cabeza del profesor, bajo el título dirigido hacia el presidente francés "*el dirigente de los infieles*", reivindicando los hechos y afirmando que: "*He ejecutado a uno de tus perros del infierno que ha osado degradar a Muhammad*" [1].

Tras la realización de averiguaciones, este no ha sido el único hecho que ha llamado la atención a los investigadores. Días antes de lo sucedido, el padre de una alumna del instituto en el que este profesor ejercía compartió en su perfil de Facebook un vídeo en el que calificaba al profesor de "sinvergüenza" debido a que este invitara a sus alumnos musulmanes a salir de clase, para posteriormente enseñar a los demás alumnos una caricatura de un hombre desnudo presentándolo como el profeta Mahoma [1].

A raíz de los actos ocurridos, a la vista está el creciente auge de este tipo de contenidos por las redes sociales, las cuales podrían llegar incluso a considerarse como el nuevo escenario para las protestas. La transcendencia que puede tener, por ejemplo, un tweet no tiene comparación con el mismo comentario realizado de viva voz, es decir, un comentario realizado entre amigos.

En este sentido, las tecnologías enfocadas al desarrollo de algoritmos de detección de odio se encuentran en plena expansión, según indica el reciente informe realizado por el Instituto de Investigación Interregional sobre Crimen y Justicia de las Naciones Unidas [2], en el cual se revela que: *"Las teorías conspiratorias que han permeado las redes sociales durante la pandemia van más allá de la idea de que las señales 5G sean las que transmitan el virus, o que el COVID-19 sea un engaño o que un billonario quiera implantarnos microchips. Grupos terroristas, extremistas, supremacistas blancos y hasta carteles mexicanos están utilizando las redes sociales para difundir su narrativa, aumentar la polarización mundial y la desconfianza en los Gobiernos, y así reclutar más adeptos para infundir más terror y odio, y ganar territorio para sus actividades ilegales."*

Es así que se ve la necesidad de lograr crear un detector automático de discurso de odio para poder filtrar este tipo de contenido antes de que llegue a convertirse en viral y desembocar en un acto de violencia, ayudando así a que este tipo de actos puedan llegar a ser erradicados.

## 1.2 Discurso de odio

Para iniciar este trabajo es necesario comenzar explicando qué se entiende por discurso de odio. El discurso de odio puede ser definido como: *"toda comunicación que menosprecie a una persona o a un grupo sobre la base de alguna característica como la raza, el color, la etnia, el género, la orientación sexual, la nacionalidad, la religión u otra característica"* [3].

Existe mucha ambigüedad acerca de la definición del término de discurso de odio, lo cual puede convertir la detección de este tipo de contenido en una tarea difícil. Además, las plataformas de redes sociales, ayudan a generar una comunicación de intolerancia que, sumada al odio, tiene un gran alcance en la sociedad y llega a ser en algunos casos incluso peligroso. Muestra de esta intolerancia se incluye dentro de la definición que apunta Weber como la más genérica [4]: *"El término Discurso de Odio se entenderá que abarca toda clase de expresión que difunde, incita, promueve o justifica el odio racial, la xenofobia, el antisemitismo u otras formas de odio basadas en la intolerancia, incluida la intolerancia que se manifiesta a través del nacionalismo agresivo y el etnocentrismo, la discriminación y la hostilidad contra las minorías, migrantes y personas de origen inmigrante."*.

La importancia de la detección del discurso de odio se centra en la relación entre este y los actuales crímenes de odio, como el ocurrido en octubre de 2020 (ver apartado 1.1). Intentar lograr una detección temprana de usuarios que promueven el odio es muy importante para prevenir acciones posteriores de violencia.

Además de esto, también hay que reseñar que existe una controversia entre lo que se denomina libertad de expresión (comúnmente conocido en la literatura como "*Free speech*") y el discurso de odio (conocido en la literatura como "*Hate speech*"). El primero se define como un intercambio de opiniones a través de debates abiertos, una forma de intercambiar, enseñar, aprender y desafiar las perspectivas de los demás; mientras que el segundo término se resume como apuntar a grupos particulares con intenciones maliciosas o insultos sobre la identidad individual, apelando contra la raza, el color, etc.


**Figura 1-1. Controversia discurso de odio/ libertad de expresión (extraído de [5]).**

Es así que, mientras el discurso de odio sigue evolucionando, a los desarrolladores de herramientas de detección de este tipo de lenguaje les resulta complicado identificar y monitorizar este tipo de contenido presente en Internet, sin dejar de separar entre lo que debería ser simplemente libertad de expresión y un ataque de intolerancia.

En cuanto al contenido de odio en las redes sociales, cabe destacar la facilidad con la que estos contenidos llegan a miles de usuarios, promovidos por los constantes avances tecnológicos. Esto supone un nuevo reto para lograr la detección del contenido de odio que se encuentra dispersado por la red, el cual no solo preocupa por su alcance, sino por su capacidad de incitación en la comunidad. Está a la orden del día que esta es la forma en la que los grupos terroristas utilizan estas plataformas para compartir sus ideas radicales y así conseguir un amplio reclutamiento.

A lo largo de la historia, el discurso de odio ha sido utilizado para atacar a personas o grupos con el único propósito de que estos sean estigmatizados e incitar así al odio y a la violencia. Así, el discurso de odio puede ser clasificado en diversos tipos, como pueden ser, relacionados con la religión, el género, la orientación sexual, el origen de donde proceden u otras características o condiciones personales. Este último es que se usó, por ejemplo, como herramienta útil en el holocausto nazi y el genocidio de Ruanda.

Este problema social es tan preocupante que incluso la UNESCO contribuyó a promover un buen uso de las redes para hacer frente al mismo publicando un estudio llamado *Countering online hate speech.*

No se puede dejar de lado la idea de que las redes son el medio perfecto para esta práctica debido a que proporcionan anonimato, con el cual todo el mundo se siente libre, sin miedo a una sanción o a la repercusión social que unas simples palabras pueden llegar a tener.

## 1.3 Objetivos del trabajo

Los objetivos que se pretenden alcanzar en el presente TFG se enumeran a continuación:

- Familiarización con las técnicas de aprendizaje de *Machine Learning*, así como una introducción al *Deep Learning*.
- Creación de varios sistemas de detección de discurso de odio en redes sociales considerando la extracción de características léxicas y no léxicas a partir de estos mensajes.
- Realización de un amplio conjunto de pruebas de los sistemas creados usando mensajes procedentes de redes sociales escritos en inglés, realizando una comparativa en función de los algoritmos utilizados y de las características extraídas.
- Realización de pruebas de los sistemas creados usando mensajes de redes sociales escritos en otro idioma, como por ejemplo el árabe.

## 1.4 Organización de la memoria

La memoria de este TFG está dividida en cinco capítulos, complementados con un anexo.

En el presente capítulo se describe una breve introducción al discurso de odio seguido de los objetivos marcados y la organización de la memoria.

En el Capítulo 2 se desarrolla el estado del arte. Se introducen las técnicas actuales para crear un detector automático de odio, introduciendo así la Inteligencia Artificial y el aprendizaje automático. Se presentan además trabajos realizados que tratan de dar solución a la detección de odio.

El Capítulo 3 se divide en cuatro partes. En primer lugar, se describen las herramientas que se utilizan para el desarrollo del TFG, seguido del preprocesado, la explicación de la extracción de

características y algoritmos empleados para crear ese detector de odio. Se finaliza con la explicación de las funciones que se implementan para llevar a cabo el entrenamiento de los algoritmos y las métricas usadas para su evaluación.

En el Capítulo 4 se expondrán los resultados experimentales obtenidos tras la realización de un amplio abanico de pruebas mediante la aplicación de numerosas características en los diversos algoritmos utilizados. Para todos ellos, se realizará una evaluación seguida de un análisis de los mismos.

Por último, en el Capítulo 5 se expondrán una serie de conclusiones tras el análisis de los resultados obtenidos en el capítulo anterior, así como posibles líneas futuras con las que se podría continuar con las pruebas desarrolladas en el presente trabajo.

El anexo incluye el código desarrollado en el presente trabajo.

# 2 ESTADO DEL ARTE

En el presente capítulo se comenzará explicando la importancia que tiene la Inteligencia Artificial para el objeto de estudio que se propone, centrando la atención en los campos de actuación del procesamiento del lenguaje natural y del aprendizaje automático, ambos de interés por su aplicación posterior en el desarrollo del proyecto.

A continuación, se hará un breve recorrido por artículos de interés relacionados con la detección de discurso de odio en redes sociales y se analizarán las soluciones propuestas para la creación de un detector automático de este tipo de contenido, recalcando las diferentes características extraídas.

## 2.1 Inteligencia artificial

La Inteligencia Artificial (IA) se centra en tratar de dotar a las máquinas de inteligencia, es decir, que sean capaces de realizar un desarrollo inteligente imitando a un ser humano. Su fin es desarrollar técnicas que permitan a las computadoras aprender sin estar explícitamente programadas para ello. Funciona combinando elevadas cantidades de datos con un procesamiento recursivo y rápido junto con algoritmos inteligentes, lo que permite que el software aprenda de forma automática patrones o características de los conjuntos de datos que se le proporciona, detectando correlaciones entre variables, para así clasificar grandes cantidades de datos y detectar diferencias y errores entre ellos. En resumen, trata de obtener análisis predictivos con fines precisos y establecer correlaciones entre varios sucesos [6]. La IA tiene diversos campos de actuación como pueden ser los que a continuación se exponen:

- Razonamiento y resolución de problemas,
- Representación del conocimiento,
- Planificación,
- Procesamiento de lenguaje natural (PLN),
- Aprendizaje (*Machine Learning* y *Deep Learning*),
- Percepción.

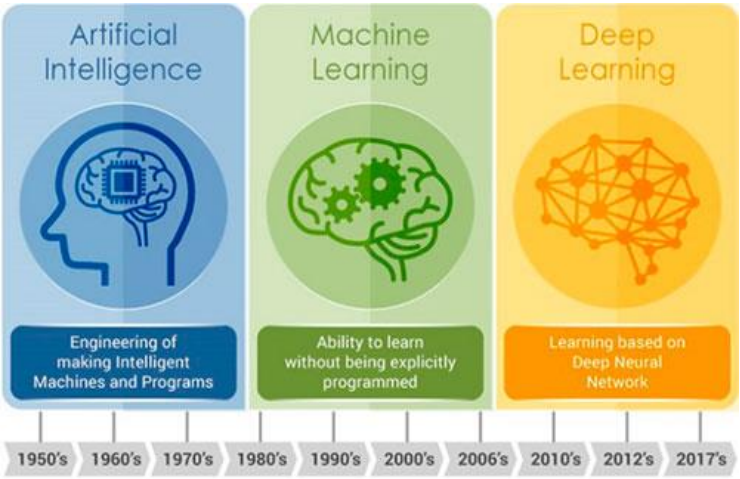


**Figura 2-1. Evolución inteligencia artificial (extraído de [6]).**

En la Figura 2-1 se muestra la evolución temporal que ha seguido la IA a lo largo de la historia. La IA engloba el aprendizaje automático (*Machine Learning*), y este a su vez engloba el aprendizaje profundo (*Deep Learning*). Sin embargo, hoy en día se entiende erróneamente que la IA está compuesta únicamente por el aprendizaje automático, y, como se ha visto anteriormente la IA está formada por más subgrupos además de este.

De entre los campos de actuación mostrados anteriormente, este trabajo se centrará concretamente en la utilización de algoritmos de aprendizaje (*Machine Learning* y *Deep Learning*), a la vez que se usarán técnicas de PLN. Debido a la importancia de ambos dentro del TFG, se van a desarrollar más en detalle en los apartados siguientes.

### *2.1.1 Procesamiento del lenguaje natural*

El PLN es el campo de conocimiento de la IA que se ocupa de investigar el modo de comunicar las máquinas con las personas a través del uso de lenguas naturales, como el español o el inglés. En definitiva, se trata de que los ordenadores analicen el lenguaje, lo interpreten y le den un significado de forma que pueda ser usado de una manera práctica. No es suficiente con entender palabra por palabra, sino que lo que interesa es conocer el sentido de las oraciones, pudiendo así realizar un análisis global de los textos que se presenten. Por tanto, se podría decir que el PLN se centra en el procesamiento de la comunicación humana, para lo cual realiza una división e identifica los elementos más relevantes del mensaje. Es importante reseñar que este procesamiento no dota de inteligencia a una máquina, solo le da la capacidad para procesar el lenguaje humano y analizarlo.

En este sentido, el PLN es ampliamente utilizado en diversas funciones y áreas, y no solo son utilizadas en chatbots, como uno podría pensar. Ejemplos de ello son:

- Comprensión del lenguaje natural, es decir, comprensión de un idioma,
- Recuperación de información,
- Reconocimiento y síntesis del habla,
- Traducción automática,
- Análisis de sentimiento o emociones.

Pero, ¿cómo se realiza este proceso? Para dar respuesta a esta pregunta es necesario explicar algunos de los componentes básicos en los que puede dividirse (Figura 2-2):

1. Análisis morfológico o "léxico": El objetivo de esta fase es identificar las unidades léxicas que componen las frases, es decir, dividir el texto a estudio en sus componentes léxicos, y asignar las etiquetas a cada una de ellas, tales como su género, número y/o persona

(llamado etiquetación). Esta última tarea es denominada como POS tagging (Part-Of-Speech tagging).
El proceso de etiquetación resulta sensiblemente más complejo que manejar un simple diccionario de palabras con su correspondiente etiqueta, ya que algunos términos pueden pertenecer a diferentes categorías dependiendo del papel que jueguen en una frase concreta. Es lo que se conoce como ambigüedad léxica.

2. Análisis sintáctico: Una vez identificadas y analizadas individualmente las palabras que componen un texto a nivel léxico, el siguiente paso consiste en establecer cómo se organizan y relacionan, así como cuál es la función de cada una, es decir, identificar la estructura sintáctica. Esta tarea es conocida como parsing.

3. Análisis semántico: El propósito aquí es identificar el significado de las oraciones, retomando los de las palabras en el contexto de su estructura sintáctica, además de que sea coherente que esta tenga sentido.

4. Análisis pragmático: Por último, este análisis identifica la conexión entre las palabras y el contexto en donde son utilizadas, debido a que este influirá en la interpretación del significado. Es necesario este tipo de análisis debido a que las frases pueden no tener su significado literal, si no que pueden ser utilizadas como una sentencia irónica.


**Figura 2-2. Proceso análisis de textos.**

Muchos de los avances en algunas tareas del PLN se deben a los logrados en la creación de algoritmos de aprendizaje automático, y más concretamente de algoritmos de aprendizaje profundo como por ejemplo las redes neuronales artificiales.

### *2.1.2 Aprendizaje automático*

El aprendizaje automático (*Machine Learning*) se define como el uso de algoritmos para encontrar patrones en los datos. Es la rama de la IA que permite el aprendizaje de las máquinas sin que estas hayan sido expresamente programadas para ello, lo cual permite hacer predicciones. Partiendo de conjuntos de datos o incluso de bases de datos, se ponen en marcha algoritmos que hacen posible realizar correlaciones entre variables, clasificar elevadas cantidades de datos y ser capaz de detectar diferencias entre ellos.

En el ámbito computacional, el aprendizaje debe ser entendido como aquello que la máquina es capaz de aprender basándose en la experiencia y no en patrones programados con anterioridad. Por tanto, la máquina realiza el proceso de aprendizaje a través de objetos con los que se entrena, aplicando posteriormente los patrones obtenidos de este reconocimiento sobre objetos diferentes. Esta tecnología se encuentra presente en el día a día, como pueden ser los sistemas de recomendación de Netflix [7] o el habla de Alexa [8].

Según explica uno de los científicos de datos del BBVA [9] *"En definitiva, el 'machine learning' es un maestro del reconocimiento de patrones, y es capaz de convertir una muestra de datos en un programa informático capaz de extraer inferencias de nuevos conjuntos de datos para los que no ha sido entrenado previamente"*.

Los algoritmos anteriormente nombrados se pueden agrupar dependiendo de los datos de entrenamiento. En el aprendizaje supervisado los resultados que se desean obtener se conocen previamente, mientras que en el caso del no supervisado el resultado no tiene porqué conocerse durante el entrenamiento. Esta clasificación se muestra a continuación:

- Aprendizaje supervisado
  Este tipo de algoritmo trabajan a partir de datos etiquetados, que sirven para entrenar el sistema, buscando una función que a partir de las variables de entrada consigan asignar la etiqueta de salida correcta. Se ajustan al conjunto de ejemplos de los que se conoce previamente la relación entre la entrada y la salida.

- Aprendizaje no supervisado
  Este por el contrario parte de datos sin etiquetas para el entrenamiento, conociendo únicamente los datos de entrada, sin existir datos de salida que se relacionen con una determinada entrada. Sólo se podrá describir la estructura de datos para realizar la búsqueda de un tipo de patrón que facilite el análisis.

Concretamente, el desarrollo de este trabajo se centrará en algoritmos de aprendizaje supervisado, siendo los más reseñables *Support Vector Machines* (SVM) y clasificación de *Naïve Bayes* (NB). Además, se experimentará en el campo del aprendizaje profundo, siendo en este caso los algoritmos destacables *Multi-Layer Perceptron* (MLP) y *Convolutional Neuronal Networks* (CNN). Todos estos algoritmos pasan a desarrollarse a continuación.

#### *2.1.2.1 Algoritmo Support Vector Machine*

Las máquinas de vector soporte (SVM, *Support Vector Machine*) son un método supervisado de clasificación que permiten realizar una separación de categorías partiendo de un conjunto de datos representados por puntos en un plano, y que pretenden encontrar un hiperplano que consiga separar los bloques de puntos. El hiperplano buscado es el denominado de máximo margen, es decir, el que posee más distancia entre bloques de datos. A estos márgenes se les denominan vectores soporte [10].

Partiendo de unos ejemplos, se entrena un SVM, separando los datos en dos categorías en base a una línea, llegando así a construir un modelo que pueda predecir la clase de una nueva muestra de datos que se le proporcione. Es necesario establecer la línea de frontera que divida el espacio para poder llegar a la clasificación de los mismos.

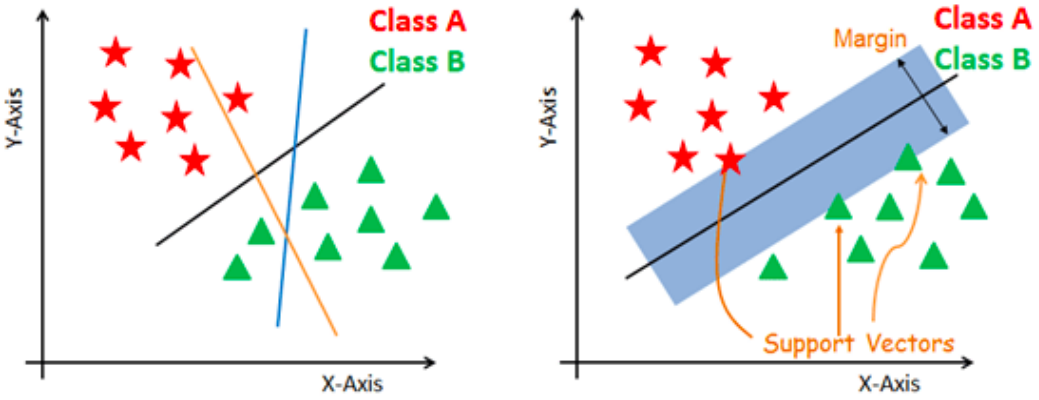


**Figura 2-3. Modelado SVM (extraído de [11]).**

### *2.1.2.2 Algoritmo Naïve Bayes*

Este método supervisado de clasificación es uno de los más utilizados y está basado en el Teorema de Bayes. Este método incluye la suposición de que existe una independencia entre sí de las características particulares en una clase y otras características. La suposición que se hace es denominada independencia condicional de clase [12].

La fórmula del Teorema de Bayes se presenta a continuación (Ecuación 1):

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}.$$

**Ecuación 1. Teorema de Bayes.**

La clasificación que realiza este algoritmo, suponiendo el caso de que solo se tenga una única característica, puede ser resumida en varios pasos:

- Cálculo de la probabilidad previa para las etiquetas de las clases dadas;
- Determinación de la probabilidad con los distintos atributos para las clases;
- Introducción de los valores en el teorema de Bayes y cálculo de P*(h|D);*
- Observación de qué clase tiene una probabilidad más alta debido a que la variable de entrada pertenece a la clase de probabilidad más alta.

Existen varios tipos de implementaciones, siendo de interés para este trabajo los algoritmos *Multinomial* [13] y *Gaussian* [14].

El primero de ellos implementa el algoritmo anteriormente expuesto para datos distribuidos multinomialmente. Para implementar el mismo en algoritmos centrados en aplicaciones de PLN, los datos han de estar representados como vectores de frecuencias de palabras o por vectores *TF-IDF*[1]. La distribución se encuentra parametrizada por vectores para cada clase "*y*" del tipo $\theta_y = (\theta_{y1}, \dots, \theta_{yn})$, siendo esta *n* el número total de características y $\theta_{yi}$ la probabilidad $P(x_i|y)$ de la característica *i* que aparece en una muestra perteneciente a la clase "$y$ ".

Los parámetros $\theta_y$ son estimados por una versión suavizada de la máxima similitud, recuento de frecuencia relativa.

Por otro lado, el segundo, denominado *Gaussian*, implementa el algoritmo a partir de la suposición de que las características tienen una probabilidad gaussiana, por lo que la fórmula a la que se llega , dada la variable de clase "*y*" y el vector de características dependientes $x_1$ a través de $x_n$, aplicando a esta última variable el supuesto ingenio de independencia condicional, es la expresión que sigue (Ecuación 2) :

$$P(x_i|\mathrm{y}) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right).$$

**Ecuación 2. Probabilidad gaussiana.**

Donde igualmente los parámetros $\sigma_y$ y $\mu_y$ son estimados por máxima similitud [15].

1 Medida numérica correspondiente a la frecuencia inversa de ocurrencia de un término en la colección de documentos. Será explicado con más detalle en el apartado 3.2.2.4 de la presente memoria.

### *2.1.2.3 Deep Learning*

El aprendizaje profundo (*deep learning*) constituye una subclase dentro del aprendizaje automático. Este se define como un algoritmo automático estructurado o, también denominado jerárquico, que simula el aprendizaje humano para obtener conocimientos específicos. La principal diferencia con el aprendizaje automático es [16]:

- Machine Learning se entiende como un subconjunto de IA donde las personas "entrenan" a las máquinas para reconocer patrones basados en datos y hacer sus predicciones. Mientras que, por otro lado, DL (Deep Learning) es un subconjunto de Machine Learning en el que la máquina es capaz de razonar y sacar sus propias conclusiones, aprendiendo por sí misma.

Cabe señalar que para que una máquina realice su proceso de aprendizaje, esta debe someterse a un proceso de enseñanza persuasivo que combine el aprendizaje supervisado y el aprendizaje no supervisado. Cuanto más cerca esté la neurona de la capa de salida, más entrenamiento supervisado será necesario para perfeccionarse. Esto sucede porque la primera capa intenta procesar lo datos para que se puedan identificar los objetos complejos, mientras que las capas más profundas requieren más atención humana porque los cálculos se vuelven cada vez más complejos.

<u>Redes neuronales</u>

Se trata de unos algoritmos y estructuras que se basan en las funciones biológicas de las redes neuronales. Se utilizan comúnmente para problemas de clasificación y regresión, y suelen obtener buenas prestaciones en la detección de patrones [17].

Las redes neuronales artificiales requieren de una gran cantidad de potencia de procesamiento y capacidad de almacenamiento, lo cual puede limitarlas debido a las tecnologías necesarias. En este sentido, existen una serie de algoritmos de interés, como el caso del perceptrón multicapa (MPL, *Multi Perceptron Layer*). Este algoritmo de aprendizaje supervisado aprende una función $f(\cdot): R^m \rightarrow R^o$ a partir del entrenamiento de un conjunto de datos, siendo *m* el número de dimensiones para la entrada y *o* el número de dimensiones en la salida. Por tanto, aportando un conjunto de características $X = x_1, x_2, \dots, x_m$ y un objetivo, aprende una aproximación de función no lineal para la clasificación o regresión. En este tipo de algoritmo puede haber varias capas entre la de entrada y la de salida, las cuales son denominadas como capas ocultas, y estas han de fijarse en la implementación del mismo.

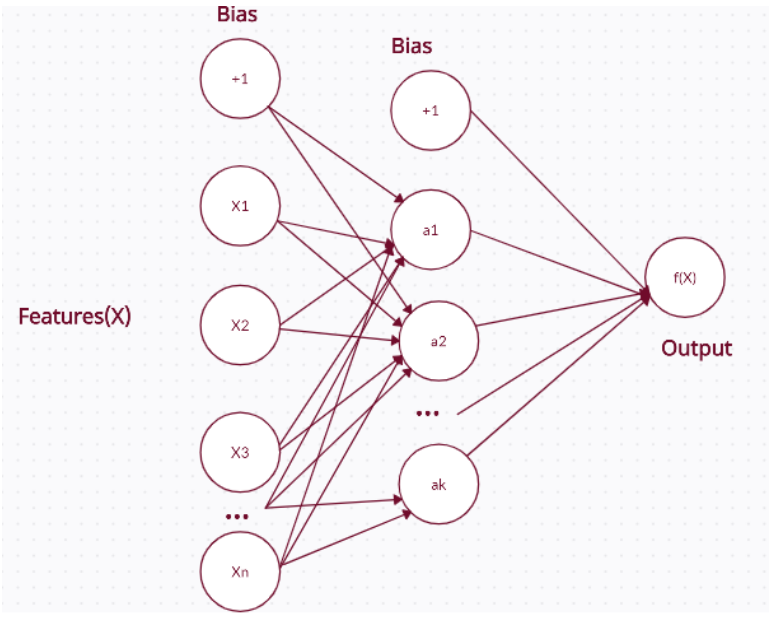


**Figura 2-4. MLP con una capa oculta.**

En la Figura 2-4, los primeros nodos que se observan se corresponden con un conjunto de neuronas, las cuales se asocian a las características de entrada. Seguidamente cada neurona de la capa oculta (se corresponde con la segunda columna) transforma los valores que obtiene de la capa anterior con una suma lineal ponderada seguida de una función de activación no lineal. Por último, la capa de salida recibe los valores de la última capa oculta y los transforma en los valores de salida.

Un ejemplo de algoritmo basado en redes neuronales que resulta de interés es el denominado red neuronal convolucional (CNN, *Convolutional Neuronal Network*). Estas redes procesan las capas imitando al córtex visual del ojo humano para que diversas características sean identificadas en los datos de entrada de manera que esta red pueda asemejar los distintos objetos que se quieran clasificar.

Las capas que esta red contiene serán:

- Capa de convolución: Procesa la salida de neuronas que están conectadas en lo que se conoce como "redes locales" de entrada, o más comúnmente llamados pixeles cercanos, calculando el producto escalar entre los pesos y una pequeña región a la que estén conectados en el volumen de la entrada.
- Capa "relu": aplica la función de activación de los distintos elementos de la matriz.
- Capa de 'pooling' o 'subsampling': Realizará una reducción en las distintas dimensiones, aunque manteniéndose la profundidad.
- Capa "tradicional": Compuesta por una red de neuronas feedforward que conecta con la última capa de la anterior nombrada y finaliza con la cantidad de neuronas que se quieren aplicar.

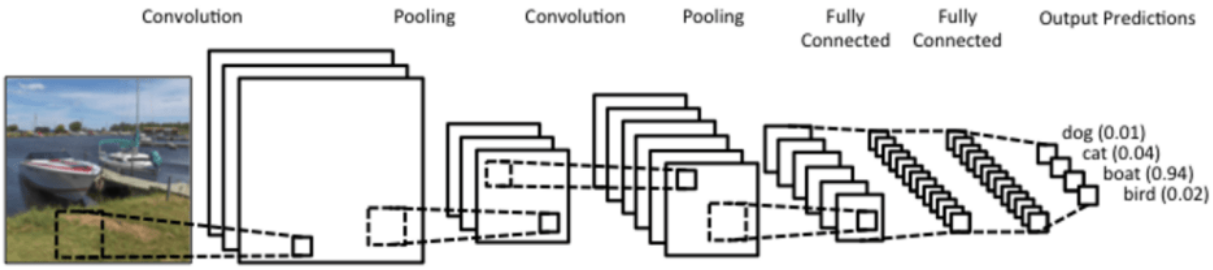


**Figura 2-5. Capas ejemplo algoritmo CNN [18].**

En la Figura 2-5 se puede observar el procesamiento de una imagen mediante CNN. La primera capa consiste en una capa convolucional, la cual se encarga de conectar cada neurona con una región específica de la imagen, aplicando una ventana de convolución o filtro. La convolución consiste en multiplicar los elementos del filtro por los elementos de la región, posición por posición para posteriormente realizar la suma de los mismos. El resultado obtenido pasará a una función de activación. La capa de pooling es utilizada para sintetizar la información proveniente de las capas anteriores mediante una operación en particular, pudiendo tomar el máximo elemento dentro de la ventana de aplicación o el promedio de los elementos dentro de la ventana. Después de repetir el proceso se llegan a las capas densamente conectadas, no usando ventana de aplicación, sino teniendo en cuenta todos los elementos del volumen, llegando finalmente a la capa de predicción.

## 2.2 Trabajos relacionados con el contenido de odio en redes sociales

A continuación, se llevará a cabo un estudio acerca de los trabajos existentes sobre la detección automática de discurso de odio. Este estudio se realizará en base a la extracción de características que se utilizan en cada uno de ellos, es decir, de la extracción en forma de vectores de datos a partir del texto. Por lo tanto, será necesario la codificación de la información para que estas puedan ser utilizadas

por los algoritmos, es decir, las informaciones obtenidas a partir de las palabras de un texto representarán características discretas y/o categóricas.

Son diversas las técnicas para expresar un texto de forma numérica. Se muestran a continuación ejemplos de trabajos anteriores con diferentes técnicas utilizadas, así como en algunos casos sus resultados, para la detección de odio.

### *2.2.1 Basado en n-gramas*

El estudio de las propiedades lingüísticas para analizar un texto puede realizarse a partir de secuencias lineales de unidades lingüísticas. Estas secuencias se conocen como n-gramas, las cuales pueden estar conformadas por 1 unidad (unigrama), 2 unidades (bigramas), 3 unidades(trigramas), etc. Las secuencias pueden ser extraídas tanto de secuencias de palabras como de secuencia de caracteres dentro de una palabra, siendo conocidos estos por ch-gramas. Un ejemplo de las secuencias mencionadas puede verse en la Figura 2-6 y en la Figura 2-7.

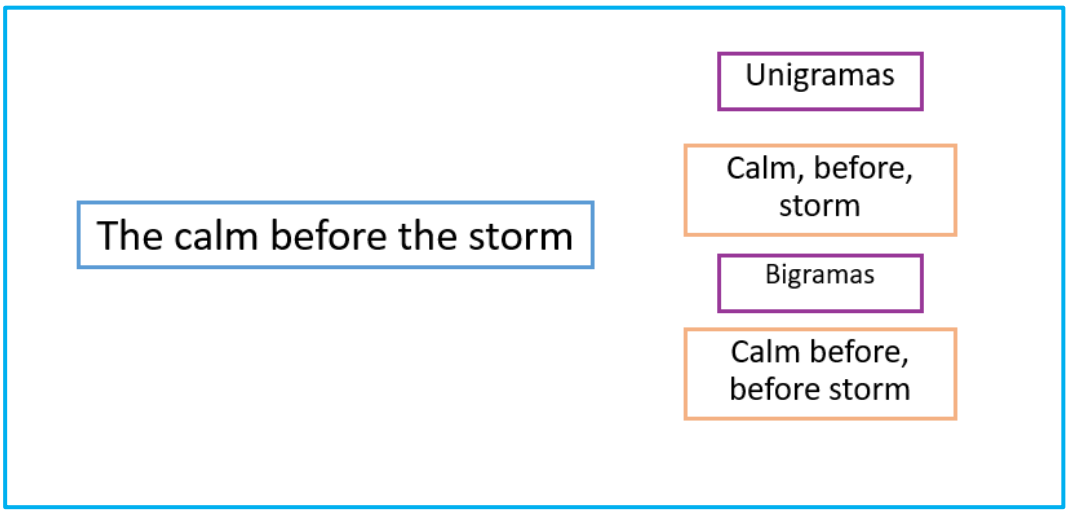


**Figura 2-6. Ejemplo analyze= Word**

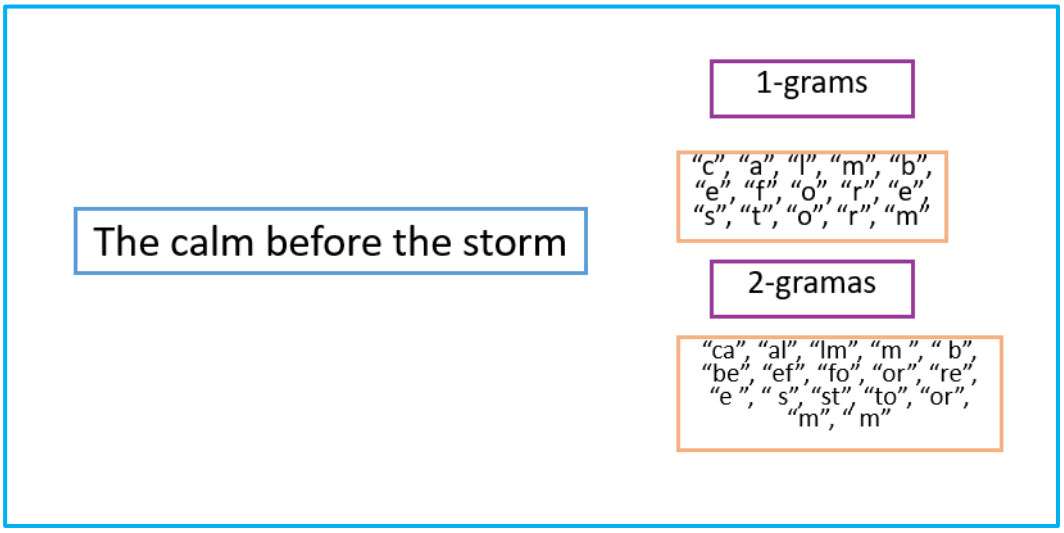


**Figura 2-7. Ejemplo analyze= Char**

### *2.2.1.1 Trigramas*

En este primer caso los autores del trabajo llevado a cabo en [19], se centraron en la detección de odio en tweets con connotación yihadista, presentando la idea principal basada en el estudio que conlleva la ubicación de una determinada palabra de odio en un texto. Es por esto que, en su investigación, Tom De Smedt muestra que, aunque una palabra aislada no sea un motivo para decir, en este caso, que se trata de un discurso de odio, la combinación de varias sí que puede ser un claro indicador.

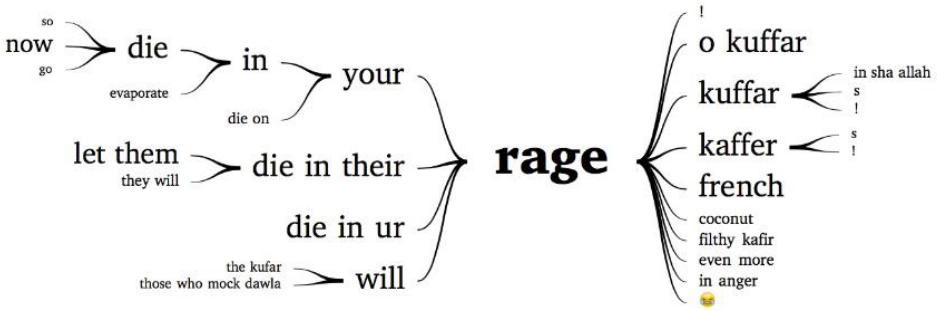


**Figura 2-8: Árbol de palabras contexto de odio de palabra "rage"(extraído de [19]).**

Así, las características elegidas para evaluar el contenido de los tweets son trigramas de caracteres (secuencias de tres caracteres sucesivos), utilizando como algoritmo en este caso SVM. De este modo, se consigue modelar eficientemente terminaciones de palabras, palabras funcionales, emojis y variaciones ortográficas.

Además de un análisis cuantitativo de la retórica yihadista, presentan un análisis cualitativo. El análisis cualitativo va referido a la situación en la cual los tweets que analizan son compartidos. Todos los mensajes recogidos fueron compartidos en la red como respuesta a ataques terroristas, aunque no se llevó a cabo un estudio de la repercusión de estos en los actos criminales que se derivaron de su publicación. Centrándose en el análisis cuantitativo, además de las técnicas de PLN comentadas anteriormente, se llevó a cabo un estudio de la relación entre los mensajes transmitidos y la edad, género, educación y personalidad de los usuarios.

### *2.2.1.2 Conjunto de n-grams*

Continuando con la técnica usada en el apartado anterior, en esta búsqueda por las características que permitan una mejor detección del odio, en [20], los autores recopilan tanto unigramas como bigramas, trigramas y 4-grams. El conjunto de datos sobre el que se centran en este caso contiene temática racistas y sexistas. Para hacer el modelo más robusto y no producir la sobreestimación del mismo realizan una validación cruzada[2] de diez capas con un algoritmo de regresión logística.

Además de utilizar estas características, y previamente al entrenamiento del modelo, realizan un análisis referido a la distribución geográfica de los tweets, distribución demográfica y al léxico.

Las conclusiones tras realizar este análisis son que las distribuciones referidas al género (dentro de la demografía) de los autores de tweets considerados de odio son en su mayoría compartidos por perfiles correspondientes a hombres, tanto aquellos referidos al racismo como comentarios respecto al género. Añaden además a su estudio la frecuencia de las palabras más utilizadas para transmitir estos mensajes de odio, encontrando que, en el caso de los tweets referidos al islam, estos sí contienen

2 Técnica para evaluar los resultados de un análisis y garantizar la independencia con la partición de los datos de entrenamiento y prueba.

palabras repetidas frecuentemente. Sin embargo, aquellos que se refieren a términos de la mujer no tienen necesariamente unos tokens que tomen más peso en el corpus.

Por otro lado, también se considera interesante conocer el lugar desde el que fueron posteados estos tweets. Por lo tanto, cuando el lugar aparece en un tweet este también se incluye como una característica. Así, su estudio refleja varias combinaciones para comprobar cuáles de ellas aportan una mayor precisión. Estas combinaciones se aprecian en la Figura 2-9:

| | char $n$-grams | +gender | +gender +loc | word $n$-grams |
|---|---|---|---|---|
| F1 | 73.89 | 73.93 | 73.62* | 64.58 |
| Precision | 72.87% | 72.93% | 72.58% | 64.39% |
| Recall | 77.75% | 77.74% | 77.43% | 71.93% |

**Figura 2-9. Combinación de características (extraído de [20]).**

- Género del usuario (gender): Utilizan desde bigramas a 4-grams e información relativa al género, y de este modo obteniendo el mayor resultado.
- Longitud[3]: Introducen en este caso la longitud total de los tweets además de los n-gramas mencionados anteriormente.
- Género + localización: Usan igualmente los n-gramas combinados con el género y la localización, pero en este caso el porcentaje de acierto del sistema sufre un decremento respecto a las pruebas anteriores.
- Género + localización + longitud: Concatenando todas las características que se propusieron disminuye aún más la precisión.

Encuentran así que el uso de n-gramas para la detección de odio es una de las mejores soluciones a seguir, aunque el uso de la localización y el género para que aporten información extra deberían mejorar la detección de este tipo de comentarios.

### *2.2.2 Basado en bolsa de palabras*

Otros autores utilizan las bolsas de palabras, ("*bag of words*"[4]) para realizar una búsqueda del odio en mensajes de Twitter, ejemplo de la formación de una bolsa de palabras se muestra en la Figura 2-10. W, Wang y Chen L. [21] vieron cómo dado el alto uso de lenguaje ofensivo, así como de "palabrotas" en las redes sociales hace que la detección de odio a través de este método sea todo un desafío. Utilizan un conjunto aleatorio de tweets basándose en este caso del acoso e incitación al odio mediante "maldiciones" publicadas en Twitter.

La utilización de las bolsas de palabras suele obtener una alta cobertura[5] en sus resultados con la aplicación de Naïve Bayes, pero, aun así, conducen a altas tasas de falsos positivos, ya que la presencia de palabras ofensivas en una oración puede llevar a una clasificación errónea de los tweets como discurso de odio, aún sin serlo.

Es importante resaltar que un alto porcentaje de las veces por las que un tweet se clasifica, por ejemplo, de racista (un 86% de los mismos según Warner W. [22]) simplemente se debe a que contiene

[3] Aunque esta característica no aparece en la Figura 2-9 también se realizan pruebas con esta característica.

[4] "*Bag of Words*" es una técnica ampliamente utilizada en PLN, la cual recoge un texto o documento en una bolsa con las palabras del vocabulario que este contiene, eliminando la gramática, pero conservando la frecuencia con la que son usadas.

[5] La cobertura (*recall*) se define la división de verdaderos positivos entre la clasificación total, es decir, relación entre los datos clasificados correctamente y la suma de todos los elementos.

palabras ofensivas. Sin embargo, hoy en día el uso de estas palabras es tan frecuente que estas se deben tener en cuenta dentro de su contexto, y no por ellas mismas aisladas del sentido de la expresión.

Se llega así a la conclusión de que, además de tener un buen contenido de términos de odio, sería necesario incluir bigramas, así como análisis de sentimiento o desambiguación de las palabras. Se habla también de una desambiguación en un modo más amplio para determinar si ese post es de odio, considerando para ello como un aspecto importante el determinar la identidad de los usuarios. De esta manera se conocería si por ejemplo "n*gga" está siendo usado como un sinónimo de "persona" o por el contrario está siendo usado como un insulto, convirtiéndose en un tweet clasificado como racista, dependiendo de la raza del usuario que escriba el comentario.

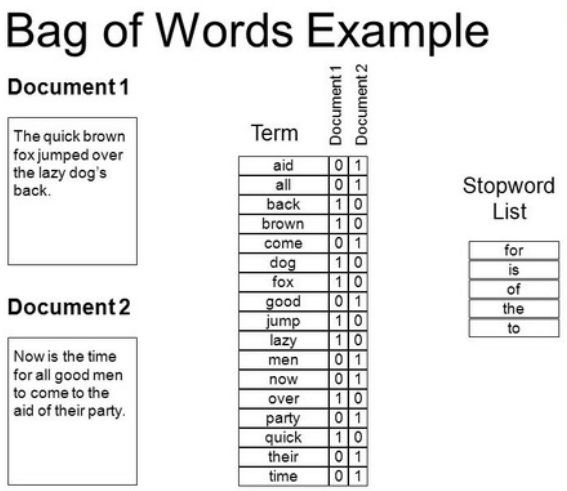


**Figura 2-10. Ejemplo de cómo se realiza un "Bag of Words" (extraído de [23]).**

Existen otros trabajos basados igualmente en bolsas de palabras. En estos casos, esta técnica para extraer características es utilizada junto con otras características para mejorar el resultado obtenido.

En [24], los autores se centran en la detección del lenguaje abusivo y detección de odio referido al sesgo racial. Para realizar su análisis, después de preprocesar[6] el texto, tokenizan y derivan estos posteriormente en n-gramas con una longitud máxima de tres. Transforman cada conjunto de datos en una matriz TF-IDF, la cual se usa también en multitud de implementaciones disponibles en repositorios de Internet, como en el caso de GitHub, para preparar los datos para su uso en un algoritmo. Entrenan su modelo con una validación cruzada de cinco estratificaciones implementando una regresión logística.

A continuación, realizan dos experimentos con los datos obtenidos anteriormente. El primer experimento trata de predecir la pertenencia a una clase, relacionando las clases con la alineación racial del idioma utilizado en cada tweet. Su objetivo es predecir la pertenencia a cada clase, siendo estas "blanco" o "negro". Quieren comprobar si la probabilidad de que se clasifiquen los tweets en base a la raza de los usuarios es igual para blancos y para negros o, por el contrario, se cumple la hipótesis alternativa que plantean los autores de que los tweets alineados con la raza negra se clasifican en su propia clase a una tasa más elevada que los tweets asociados a la raza blanca.

En el segundo experimento, evalúan si existe un perjuicio racial en aquellos tweets que contengan asociadas palabras clave relacionadas con una clase negativa. Con esta prueba tienen en cuenta que los resultados que obtuvieron en el primer experimento pueden estar influenciados por las diferencias de uso de palabras asociadas a las diferentes clases.

6 Aplicar transformaciones a los datos para que puedan ser interpretados por los algoritmos del aprendizaje automático. Este término se desarrollará en profundidad en el apartado 3.2.13.2.1.

Analizando los resultados, los autores llegan así a la conclusión de que muchos de los tweets clasificados como discurso de odio, sexistas, acoso y abuso han sido la mayoría de ellos escritos por personas de color. Esta clasificación puede deberse a la diferencia del vocabulario usado por los distintos hablantes relacionado con su forma de hablar o dialecto. Esto es, teniendo en cuenta el vocabulario empleado, se ve que estas personas suelen utilizar frecuentemente palabras de tipo informal (slang) con connotaciones negativas, como n*gga, las cuales se asocian con una clase negativa, debiendo por el contrario ser clasificadas según su sentido dentro de la oración.

Los resultados del segundo experimento refuerzan los obtenidos en el primer caso, aunque existen algunas diferencias. Siguen existiendo disparidades raciales, aunque, en cambio, son menores en magnitud que las anteriores. Algunas de las disparidades, según los autores, probablemente se deban a diferencias en las distribuciones de otras palabras clave que no tuvieron en cuenta en su análisis. Además, los valores desproporcionados de clasificación de los tweets afroamericanos, clasificando estos como negativos en su mayoría, pueden estar relacionados con etiquetas negativas que se asocian al lenguaje utilizado por la raza negra en el conjunto de entrenamiento.

### *2.2.3 Basados en reducción de las palabras a su raíz*

En [25], los autores queriendo mejorar la detección basada en bolsa de palabras, incluyen en las características para su algoritmo de detección la reducción de las palabras contenidas en los tweets a su raíz o stem usando para ello la librería *Porter stemmer* incluida en el conjunto de herramientas NLTK[7]. Posteriormente, mediante la utilización de otras librerías incluidas en NLTK construyen unigramas, bigramas y trigramas ponderados por su TF-IDF. El tipo de datos sobre el que trabajan en este caso tiene temática racista y homofóbica.

Por otro lado, añadido al TF-IDF, tomando un léxico de sentimiento diseñado específicamente para un entorno de redes sociales, tienen en cuenta la presencia o ausencia de estas palabras de forma a asignarle una determinada polaridad a cada tweet. Además, sumado a esto, utilizan indicadores binarios de presencia/ausencia de hashtags, menciones, retweets, además de su conteo y, número de caracteres, palabras y sílabas en cada tweet.

Lo que pretenden conseguir así es separar el discurso de odio en las redes sociales de otro lenguaje ofensivo, ya que trabajos anteriores no logran hacer la distinción entre estas dos categorías. Por este motivo, y utilizando crowdsourcing, etiquetan todos los tweets que lograron extraer de la red en tres categorías: odio, aquellos que no tienen lenguaje de odio y aquellos que solo incluían lenguaje ofensivo. Entrenando una máquina con las características elegidas anteriormente para clasificar, obtuvieron mejores resultados con modelos de regresión logística y SVM linear que con otros.

Llegan, por tanto, a la conclusión de que tanto los tweets racistas como los homofóbicos tienen más probabilidades de ser clasificados como discurso de odio y, por el contrario, los sexistas generalmente tienden a ser clasificados como ofensivos. Los tweets con las mayores probabilidades de ser clasificados como discurso de odio tienden a contener múltiples expresiones raciales, comentarios homofóbicos o insultos.

A la vista de este trabajo se concluye que la detección de odio en aquellos tweets que no contengan palabras clave de odio no se resuelve fácilmente.

### *2.2.4 Basados en el estereotipo*

Analizando este caso en [26], los autores utilizaron la hipótesis de que a menudo se utilizan estereotipos en los discursos de odio con el único fin de menospreciar a un individuo o grupo de

[7] NLTK es una herramienta para crear programas Python que funcionen con datos del lenguaje humano.

individuos. Para aplicar su teoría utilizan un conjunto de textos los cuales contienen expresiones regulares relativas al judaísmo e Israel. Cada estereotipo tiene un lenguaje propio con diferentes conceptos, metáforas, frases, epítetos de una palabra o yuxtaposiciones que pueden transmitir intenciones de odio. Con esto, se consigue que algunos de los textos que son evaluados sean clasificados como odio, aunque ninguna palabra lo exprese por sí misma en el pasaje.

Con esta suposición favorecen el hecho de que un texto pueda ser falsamente clasificado como no perteneciente a la categoría de discurso de odio porque ninguna palabra del mismo incita a ello, aunque, como intentan aplicar estos autores, siguiendo el estereotipo del texto podría identificarse distintas formas de discursos. Es así que, en este caso, el problema de detección del discurso de odio trata de adaptarse a un problema de desambiguación del sentido de las palabras.

Para llevar a cabo este método, las características valoradas en los textos para llegar a esta desambiguación de la que hablábamos anteriormente serán las expuestas en la siguiente imagen (Figura 2-11):

| | |
|---|---|
| unigram | ”W+0:america” |
| template literal | ”W-1:you W+0:know” |
| template literal | ”W-1:go W+0:back W+1:to” |
| template part of speech | ”POS-1:DT W+0:age POS+1:IN” |
| template Brown sub-path | ”W+0:karma BRO+1:0x3fc00:0x9c00 BRO+2:0x3fc00:0x13000” |
| occurs in ±10 word window | ”WIN10:lost W+0:war” |
| other labels | ”RES:anti-muslim W+0:jokes” |

**Figura 2-11. Ejemplo de plantillas de características (extraído de [26]).**

Detrás de este proceso de desambiguación está la distribución desigual de las colocaciones[8] respecto al token ambiguo que se esté clasificando, siendo una explicación básica de la plantilla que anteriormente se expuso la siguiente, la cual fue creada por Yarowsky [27]:

- Word immediately to the right (+1 W)
- Word immediately to the left (-1 W)
- Word found in $\pm k$ word window[5] (±k W)
- Pair of words at offsets -2 and -1
- Pair of words at offsets -1 and +1
- Pair of words at offsets +1 and +2

**Figura 2-12. Leyenda análisis tokens (extraído de [27]).**

Según esta leyenda, lo que Yarowsky quería conseguir era detectar el acento de determinadas palabras, o tokens, las cuales resultaban ambiguas, descubriendo así el tono con las que las mismas eran usadas a lo largo de un texto.

Finalmente, además del uso de unigramas, se usaron tanto bigramas como trigramas, para entrenar el algoritmo SVM utilizado, lo cual degradó el clasificador en lugar de mejorarlo, según los resultados obtenidos por los autores, aunque, lo que sí se consiguió fue modificar el problema de detección de odio convirtiéndolo en uno de clasificación. Por lo tanto, se llega a la conclusión de que son necesarias algunas mejoras en la elección de características usadas en el clasificador.

[8] Ubicación de una palabra de interés respecto a las demás palabras de una frase.

# 3 DESARROLLO DEL TFG

En este capítulo se describirán los recursos empleados en el TFG, se explicarán los pasos a seguir para extraer información de interés para realizar la detección de odio, y, se expondrán todas las funciones y algoritmos utilizados para llevar a cabo esta clasificación entre lenguaje ofensivo o no antes de pasar a validar y probar el sistema con datos acorde a los objetivos.

## 3.1 Herramientas utilizadas

Para el desarrollo del presente trabajo se necesitó en un primer momento preparar el entorno de trabajo con las herramientas que se describen a continuación.

### *3.1.1 Python*

Hoy en día, este lenguaje de programación es uno de los más utilizados a nivel global. Se encuentra entre los primeros cinco puestos del ranking TIOBE[9] (Figura 3-1), estando ubicado por encima de lenguajes como Java o C++.

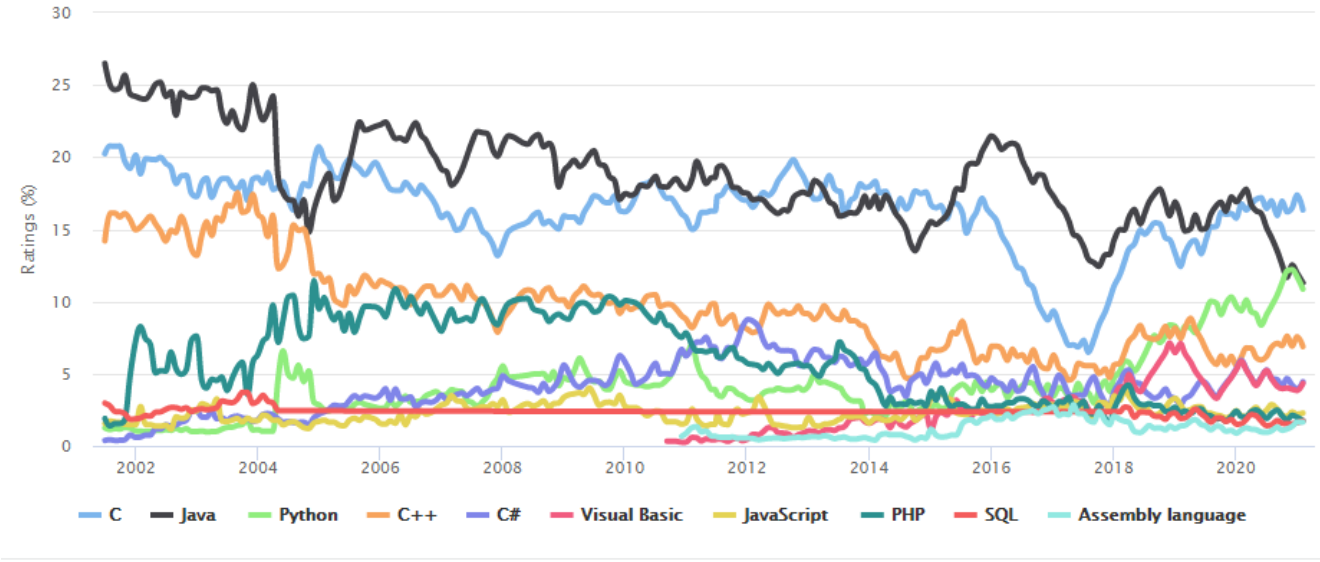


**Figura 3-1. Índice de la comunidad de programación TIOBE (extraído de [28]).**

[9] El índice TIOBE es un indicador que utiliza distintas variables como el número de personas que lo utilizan o los cursos que existen para determinar la popularidad de los lenguajes de programación.

Dispone de un sistema robusto, pero a la vez sencillo con el que resulta relativamente fácil la manipulación y gestión de cadenas, números, documentos, etc. Es un lenguaje interpretado, es decir, no necesita de la compilación del código fuente para realizar su ejecución, por lo que ofrece diversas ventajas como pueden ser la rapidez de desarrollo.

Este lenguaje de programación posee la ventaja de tener una gran cantidad de librerías externas que pueden utilizarse para realizar la implementación de modelos de *Machine Learning* de una manera más simple que con la ausencia de estas, sin tener que comenzar el código desde 0.

### *3.1.2 Miniconda*

Miniconda es una versión más pequeña derivada de Anaconda, siendo esta última una distribución de Python que lo instala, además de permitir la instalación de paquetes como los especializados en Data Science y Deep Learning.

En su caso, Miniconda, contiene únicamente los paquetes de Python, `pip` y `conda`: ambos son lo que se denominan "`package managers`", es decir, permiten descargar e instalar paquetes de manera automática. De esta forma, si necesitamos instalar algún paquete que no tenga por defecto, se realizará como se plasma a continuación en la Figura 3-2 de una forma sencilla:

```
(base) C:\Users\sheil>pip install textblob
Collecting textblob
  Downloading textblob-0.15.3-py2.py3-none-any.whl (636 kB)
     |████████████████████████████████| 636 kB 437 kB/s
Requirement already satisfied: nltk>=3.1 in c:\users\sheil\miniconda3\lib\site-packages (from textblob) (3.5)
Requirement already satisfied: tqdm in c:\users\sheil\miniconda3\lib\site-packages (from nltk>=3.1->textblob) (4.51.0)
Requirement already satisfied: click in c:\users\sheil\miniconda3\lib\site-packages (from nltk>=3.1->textblob) (7.1.2)
Requirement already satisfied: regex in c:\users\sheil\miniconda3\lib\site-packages (from nltk>=3.1->textblob) (2020.11.
13)
Requirement already satisfied: joblib in c:\users\sheil\miniconda3\lib\site-packages (from nltk>=3.1->textblob) (1.0.0)
Installing collected packages: textblob
Successfully installed textblob-0.15.3
```

**Figura 3-2. Instalación de paquetes.**

Los paquetes que, en este TFG, fueron necesarios de instalar son los que a continuación se enumeran:

- Pandas: Librería de Python especializada en el manejo y análisis de estructuras de datos [29].
- Scikit-learn: Módulo de Python para aprendizaje automático construido sobre Scipy. [30].
- NLTK: Paquete para el procesamiento del lenguaje natural [31].
- Numpy: Biblioteca que permite crear vectores y matrices grandes multidimensionales [32].
- Matplotlib: Biblioteca completa para crear visualizaciones estáticas e interactivas [33].
- Keras: API de aprendizaje profundo, ejecutada sobre TensorFlow [34].
- Tensorflow: Plataforma de aprendizaje automático desarrollado por Google [35]
- VaderSentiment: Herramienta de análisis de sentimientos [36].
- Emoji: Biblioteca emoji para Python [37]
- Unicodedata: Módulo de acceso a la base de datos de caracteres Unicode [38].
- TextBlob: Biblioteca de procesado de datos textuales, incluyendo análisis de sentimiento [39].

### *3.1.3 Visual Studio Code*

Visual Studio Code [40] es un editor de código optimizado con soporte para operaciones de desarrollo como depuración, ejecución de tareas y control de versiones. Como objetivo principal tiene proporcionar las herramientas que los desarrolladores necesitan para un ciclo rápido de código-build-debug y, por otro lado, deja flujos de trabajo más complejos para IDEs más completos, como Visual Studio IDE.

Por último, lo destacable de Visual Studio Code es que tiene una sintaxis muy visual, ya que para realizar la programación tiene una notación con márgenes implementados de obligado cumplimiento. De esta forma todos los programadores adoptaran una misma notación, teniendo por tanto todos los programas aspectos similares.

## 3.2 Implementación del sistema de detección de odio

El entrenamiento de un algoritmo de IA centrado en la detección de odio en contenido textual necesita de la realización de una serie de tratamientos de los datos, con el único objetivo el elegir las características adecuadas.

Para llevar a cabo esta extracción de características se seguirán los pasos que se citan en la Tabla 3-1, para posteriormente explicarlos con más detalle:

| | |
|---|---|
| Detección de odio | Preprocesado |
| | Conversión de datos/ Extracción de características |
| | Entrenamiento algoritmos |
| | Obtención resultados |

**Tabla 3-1. Metodología.**

### *3.2.1 Preprocesado*

Para realizar el entrenamiento de un algoritmo de aprendizaje automático, es necesario preprocesar en un primer momento el texto. Para ello, se crea una función de Python cuyo objetivo sea la eliminación del ruido de los tweets y su normalización. Este paso se aplicará siempre antes de extraer las características de los textos. La normalización podría definirse como el proceso de transformación del texto para la obtención de una forma unificada. Para ello, se han realizado las siguientes tareas, las cuales pueden observarse en la Figura 3-3:

- Conversión a minúsculas
- Eliminación de saltos de página
- Eliminación “rt” debido a que no aportan información

- Eliminación de URL´s
- Normalización puntuaciones y tildes

Se mantendrán tanto los hashtags, menciones y emojis los cuales serán utilizados como características en pasos posteriores.

```
16  def preprocess_text(s):
17      punctuation = ["!", "¡", "?", "¿", ".", "(", ")", "'",
18                     ",",">", "<",":", "/", "º", "%", "£",
19                     "€", "$", "f-","]","[", "‘", "’", "“", "”",
20                     "«","»", "=", "*", ";","&","+","_", "|",
21                     "p-","s-","-", "..."]
22      tildes = [("á", "a"), ("é", "e"), ("í", "i"), ("ó", "o"),("ú", "u")]
23      s = s.lower()
24      s = re.sub('\n', '', s)
25      s = re.sub('rt', '', s)
26     # s = re.sub(r'@[\w_]+', '', s)  #eliminar usuarios(PROBAR SIN ESTO)
27      s = re.sub(r'https?://[^\b]+', '', s) #eliminamos urls
28      #Quitar tildes
29      for j in range(len(tildes)):
30          s = re.sub(re.escape(tildes[j][0]), re.escape(tildes[j][1]), s)
31      #Quitar caracteres
32      for j in range(len(punctuation)):
33          s = re.sub(re.escape(punctuation[j]), '', s)
34      s = "".join(s)
35      #s=tokenizen(s)
36      return s
```

**Figura 3-3. Función de preprocesado.**

El módulo utilizado para el realizar el preprocesado anteriormente expuesto es el denominado `re`, el cual cuenta con las funciones apropiadas para trabajar con expresiones regulares y cadenas. La función con la que se lleva a cabo es `sub()`, con la que busca y sustituye cadenas a partir del formato que se muestra a continuación.

$$sub(cadena\ para\ sustituir, sustitucion\ de\ cadena, cadena\ donde\ se\ busca)$$

En el preprocesado no se llega a utilizar la lematización, ni la etiquetación, debido al aporte léxico que proporcionan al análisis de los tweets para la predicción que se llevará a cabo.

### *3.2.2 Extracción de características*

Los algoritmos utilizados para realizar el entrenamiento de *Machine Learning*, así como su evaluación, necesitan de la extracción de características de los textos, para posteriormente transformarlas a una representación numérica. Estos algoritmos esperan como entrada una matriz bidimensional, donde columnas se corresponderán con las características y las filas serán las instancias. Para realizar el aprendizaje automático se transformarán los documentos en representaciones vectoriales.

La representación numérica de los documentos dota de la capacidad de realizar un análisis significativo además de crear las instancias sobre las que operará el algoritmo. En el caso de estudio, las instancias serán el corpus de tweets del que dispondremos. Cada característica extraída de los

mismos equivale a cada propiedad de la representación vectorial. Se llega finalmente a un conjunto multidimensional en el que se pueden aplicar métodos de aprendizaje automático.

#### *3.2.2.1 Función extracción características superficiales*

Se comienza extrayendo características superficiales, como son el número de caracteres por cada palabra del total del tweet preprocesado (`num_char`), el número de caracteres en los tweets completos, teniendo en este caso en cuenta el tweet sin preprocesar, y también incluyendo además los espacios entre cada palabra (`num_char_total`), el número de términos (`num_terms`) y, por último, el número de caracteres por palabra en el tweet preprocesado (`num_words`).

#### *3.2.2.2 Función extracción de polaridad y sentimiento*

A continuación, se lleva a cabo una extracción de características relacionadas con la polaridad de los tweets, con funciones como `VaderSentiment` y `TextBlob`.

Se aplica el módulo de análisis de sentimientos llamado `VaderSentiment` [41], el cual está basado en reglas y léxico obtenido específicamente de los sentimientos comunes que se expresan en redes sociales.

```
def sentiment_analyzer_scores(sentence):
    score = analyser.polarity_scores(sentence)
    print("{:-<40} {}".format(sentence, str(score)))
```

**Figura 3-4. Función obtención polaridad sentencia (extraído de [41]).**

Los `polarity_scores(text)` (ver Figura 3-4) se utilizan para el análisis de los datos de texto, ofreciendo una salida en formato de diccionario en el que la clave es la etiqueta y el valor es la puntuación de polaridad.

$$\{'neg': 0.0, 'neu': 0.592, 'pos': 0.408, 'compound': 0.7345\}$$

**Figura 3-5. Diccionario valores ejemplo polarity_scores.**

Los valores situados en la Figura 3-5, desde la primera a la tercera posición representan el porcentaje de negativo, positivo y neutral que contiene la frase analizada. Los valores que obtenemos en el cuarto término se corresponden con el valor compuesto, el cual será el utilizado en la matriz de características. Estos valores están incluidos en el intervalo [-1,1], correspondiéndose -1 para negativo, 0 para neutral y 1 para positivo.

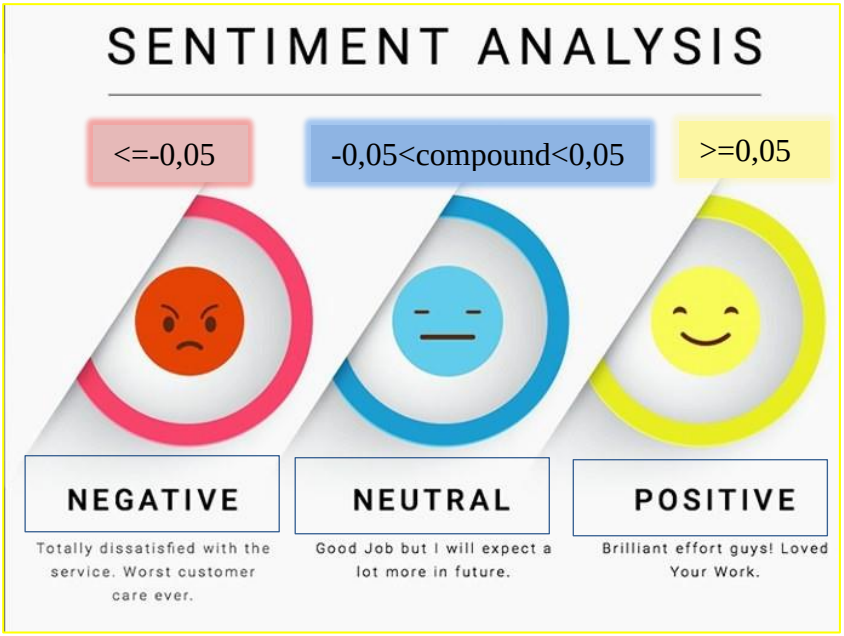


**Figura 3-6. Análisis de sentimiento (extraído en [42]).**

El rango que comprende esta función puede dar puntuaciones negativas, las cuales pueden presentar problemas al usarlos en el entrenamiento de alguno de los algoritmos, como puede ser el MultinomialNB, con lo cual, la solución adoptada será mover el intervalo que la función devuelve a valores positivos, es decir, el rango pasará a comprender un rango de [0,2].

Únicamente se analiza la polaridad de los textos, sin tener en cuenta el sentido irónico o sarcástico que, en algunos casos, se puede llegar a observar. Para poder llevar a cabo el estudio de este tono de sarcasmo y comprobar, además, que análisis de sentimiento se adapta más al corpus sobre el que se aplica, se implementa un segundo analizador de sentimiento, en este caso, `TextBlob` [39].

Este analizador permite tener en cuenta tanto la polaridad, como la subjetividad de las frases. La subjetividad es referida a la opinión personal, encontrándose los valores que devuelve entre el rango de [0,1]. Los valores se interpretan de forma que, si el valor que devuelve es superior a 0,5 la oración será más subjetiva que objetiva.

Por otra parte, la salida de la polaridad estará a su vez recogida en el rango de [-1,1] siendo -1 una polaridad negativa y +1 positivo, como se puede observar en la Figura 3-7, igual que los resultados obtenidos con `VaderSentiment`. Si el valor devuelto es igual a 0 este tweet será neutral.

`TextBlob` difiere de `VaderSentiment` en que se basa en reglas para encontrar la polaridad, tiene en cuenta la frecuencia de las palabras y relaciones semánticas, por lo que se podría obtener con `TextBlob` una salida más precisa.

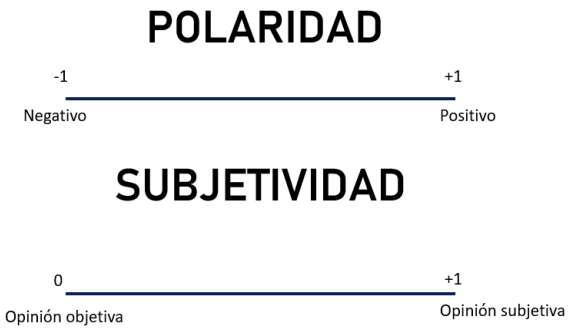


**Figura 3-7. Rangos de clasificación.**

### *3.2.2.3 Función conteo elementos lenguaje Twitter*

Teniendo en cuenta la naturaleza de los textos publicados en Twitter se implementa, además, una función para determinar la frecuencia con que se utilizan tanto hashtags como menciones en los distintos tweets. Al final, para lograr un análisis más sencillo se sustituyen los valores por 'MENTIONHERE' y 'HASHTAGHERE' y realizar así un conteo de las veces que estas son utilizadas (Figura 3-8).

```
def count_twitter_objs(text_string):
    space_pattern = '\s+'
    mention_regex = '@[\w\-]+'
    hashtag_regex = '#[\w\-]+'
    parsed_text = re.sub(space_pattern, ' ', text_string)
    parsed_text = re.sub(mention_regex, 'MENTIONHERE', parsed_text)
    parsed_text = re.sub(hashtag_regex, 'HASHTAGHERE', parsed_text)
    return(parsed_text.count('MENTIONHERE'),parsed_text.count('HASHTAGHERE'))
```

**Figura 3-8.Función conteo hashtag y menciones.**

Los tweets, además de menciones y hashtags, contienen emojis, de los cuales se puede inferir un sentimiento positivo, negativo o neutral, por lo que se considera interesante incluir estas polaridades en la extracción de características.

Para implementar esta función se crea un diccionario a partir de un diccionario de emoji[10] el cual contenía en una columna las polaridades correspondientes a otra columna de emojis en su código UNICODE. Se comparan los tweets con este diccionario sustituyendo los mismos por @N@, @NONE@, @P@, a los cuales se les llevará a cabo un conteo posterior. Al final, se obtendrá como resultado el número de emojis positivos, neutrales y negativos presentes en los tweets. La función implementada se puede observar en la Figura 3-9.

```
52  #Creacion de diccionario
53  df_annotated=pan.read_csv('csv\R_annotated_en.csv')
54  df_annotated_uni=df_annotated['UNICODE'].values.tolist()
55  df_annotated_tag=df_annotated['TAG'].values.tolist()
56
57  dicEmoji= dict(zip(df_annotated_uni,df_annotated_tag))
58
59
60  def count_emoji_objs(text_string):
61      space_pattern = '\s+'
62      data = regex.findall(r'\X', text_string)
63      text_string2=text_string
64
65      for Word in data:
66          if any(char in emoji.UNICODE_EMOJI_ENGLISH for char in Word):
67              if len(Word)<=1:
68                  s='U+{:X}'.format(ord(Word))
69                  if s in dicEmoji:
70                      text_string2=re.sub(Word,' @'+dicEmoji[s]+'@ ',text_string2)
71
72      return(text_string2.count(' @NONE@ '),text_string2.count(' @P@ '),text_string2.count(' @N@ '))
73
```

**Figura 3-9.Función conteo emojis.**

[10] Diccionario obtenido del artículo [50].

### *3.2.2.4 Función TF-IDF*

La representación del documento como un vector cuya longitud es igual al vocabulario del corpus se lleva a cabo mediante un enfoque de bolsa de palabras. Son abundantes las técnicas para la codificación virtual, como pueden ser one-hot, TF-IDF, o representaciones distribuidas. Estas técnicas pueden ser implementadas tanto en Scikit-Learn, Gensim[11] y NLTK [43].

La técnica utilizada en este caso será TF-IDF (*term frecuency-inverse document frecuency*). La elección de esta frente a las citadas anteriormente se debe al enfoque que esta ofrece al tener en cuenta todo el contexto del corpus que se le muestra para crear una representación de bolsa de palabras. Ofrece una perspectiva más interesante en la que en lugar de buscarse las palabras más frecuentes, se considera la frecuencia relativa, es decir, se centra en las palabras más extrañas del corpus, aquellas en las que es más probable que se encuentre el significado que quieren transmitir los usuarios en los tweets [44].

Se llega, por tanto, a la normalización de la frecuencia de los tokens en un documento con respecto al resto del corpus, acentuando los términos que se consideran relevantes para una instancia específica, como se puede observar en la Figura 3-10, donde los tokens de mayor relevancia son los que menos aparecen.

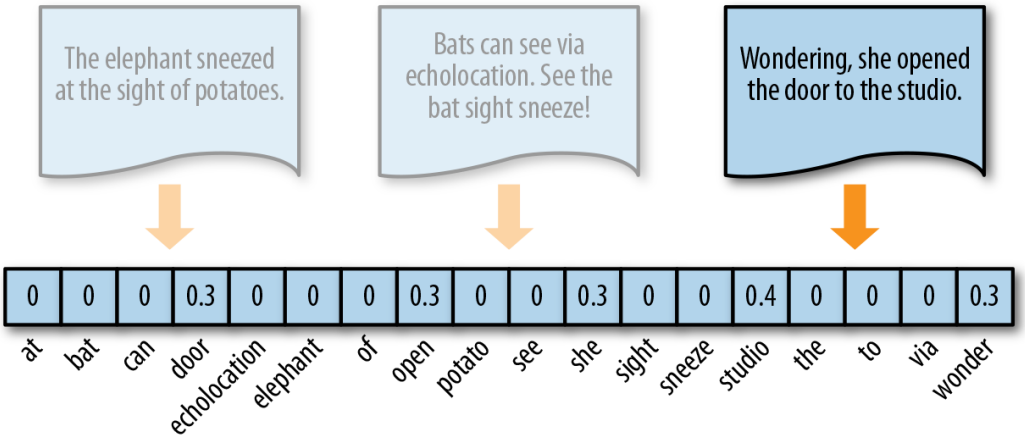


**Figura 3-10. Distribución pesos TF-IDF (extraído en [43]).**

La transcripción matemática en la que se basa TF-IDF es la que se observa en la Figura 3-11:

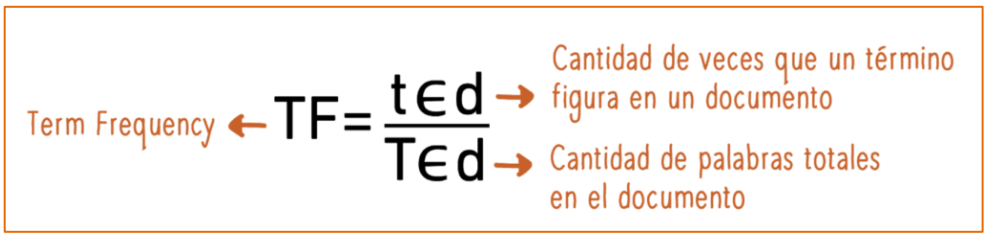


**Figura 3-11. Fórmula TF (extraído en [45]).**

[11] Biblioteca de Python para modelado de temas, indexación de documentos y recuperación de similitudes con grandes conjuntos de datos [49].

Se conoce a partir de esta fórmula la frecuencia del término, realizando este cálculo para cada palabra de cada documento según la fórmula de la Figura 3-12.


**Figura 3-12. Fórmula IDF (extraído en [45]).**

Por otro lado, se obtiene así el valor de la frecuencia inversa, llegando a determinar el valor de TF-IDF como multiplicación de ambas fórmulas (Figura 3-13).

$$TF\text{-}IDF = TF \times IDF$$

**Figura 3-13. Fórmula matemática TF-IDF (extraído en [45]).**

La interpretación de la puntuación debe entenderse como: cuanto más cercana a 1 sea la puntuación del término, más información aportará ese token para el documento; y, por el contrario, cuanto más cercano a 0 menos información dará.

Utilizando la librería de Scikit-Learn se implementa `TfidVectorizer` para vectorizar el documento con valores obtenidos de la puntuación TF-IDF.

`TfidfVectorizer` utiliza además `CounterVectorizer` para producir la codificación de la bolsa de palabras, y llevar a cabo el conteo de las ocurrencias de los distintos tokens.

Se tienen varias opciones de extracción de las frecuencias de los tokens, analizando caracteres o en su defecto palabras. Para llegar a la solución del problema se utilizarán ambas. Así, primeramente, se aplica un método de tokenización, por el mismo `TfidfVectorizer`, analizando posteriormente tanto caracteres como palabras.

Para no saturar el sistema, debido a que se encuentra un problema de memoria necesaria en la máquina de trabajo, se delimitan las frecuencias poniendo un mínimo de 50 y un máximo de 700000, eliminando así también los términos menos interesantes para el análisis. Añadido a estos límites establecidos, se suprimen además las palabras vacías de significado, comúnmente conocidas como stopwords. Las palabras vacías son eliminadas para evitar que se interpreten como una señal de predicción. Estas palabras vacías son específicas para cada idioma.

A este respecto, `TfidVectorizer` puede tratar directamente con las palabras o con los caracteres, como se puede observar en la Figura 3-14. En este trabajo se ha optado por usar ambos, considerándolas como características diferenciadas. En este sentido, se usan tanto bigramas como trigramas para los dos casos. Se muestra en el apartado 2.2.1 un ejemplo del proceso de extracción de unigramas y bigramas en un tweet.

```
27  #TfidfVectorizer
28  tfi_df=TfidfVectorizer(sublinear_tf=True,analyzer='word', norm='l2',
29                         encoding='latin-1', ngram_range=(1, 2), stop_words='english', min_df=50,max_df=700000)
30
31  tfi_df2=TfidfVectorizer(sublinear_tf=True,analyzer='char', norm='l2',
32                         encoding='latin-1', ngram_range=(2, 2), stop_words='english', min_df=50,max_df=700000)
```

**Figura 3-14. TfidVectorizer palabras(word) y caracteres(char).**

Posteriormente a este análisis, el vectorizador devuelve una representación matricial dispersa con la puntuación TF-IDF.

### *3.2.2.5 Función CountVectorizer*

En el caso de esta función se convierte el corpus de datos en una matriz en la que cada una de las palabras del texto se corresponde con una columna cuyo valor será el número de veces que dicha palabra se encuentra en cada tweet.

De esta forma se podrán realizar pruebas para meter como características a los algoritmos el conjunto de unigramas, bigramas y, por otro lado, bigramas y trigramas de caracteres.

Se combinará esta matriz obtenida con las características superficiales que se comentaron en el apartado 3.2.2.1, para comprobar su precisión, al añadir a las características más básicas, características léxicas, sin tener en cuenta en este caso la frecuencia inversa como hacia `TfidVectorizer`.

### *3.2.2.6 Concatenaciones características*

Una vez se poseen todas las características expresadas anteriormente es necesario concatenarlas en un array conjunto. Se realizan dos concatenaciones: una para el conjunto de características superficiales, análisis de sentimiento y conteo de elementos propios del lenguaje de Twitter, es decir, características numéricas, función `get_features`; y, la segunda de ellas para la concatenación de la anterior con las características léxicas proporcionadas por `TfidfVectorizer` y `CounterVectorizer`, función `transform_inputs`.

#### ***3.2.2.6.1 Función get_features***

Las características numéricas se concatenan en un array, pudiendo realizar combinaciones de las mismas según las características que se quieran extraer de los textos en el array denominado `featurs` (Figura 3-15).

```
84  def get_features(s):
85
86      data_full = s                    #datos completos
87      data = preprocess_text(s)        #datos preprocesados
88
89      #Caracteristicas superficiales
90      num_chars=sum(len(w) for w in data)
91      num_chars_total=len(s)
92      num_terms=len(data_full)
93      num_words=len(data)
94
95      #analisis de sentimiento
96      sentiment = sentiment_analyzer.polarity_scores(s)
97      analysisPol = TextBlob(s).polarity
98      analysisSub = TextBlob(s).subjectivity
99
100     #Elementos lenguaje Twitter
101     twitter_objs = count_twitter_objs(data_full) #Count #, @
102     emojis=count_emoji_objs(data_full)
103
104     #Concatenacion
105     featurs=[num_terms,num_chars_total,num_words,num_chars, sentiment['compound']+1,analysisPol+1,analysisSub,
106             twitter_objs[0],twitter_objs[1], emojis[0],emojis[1],emojis[2]]
107
108     return featurs
```

**Figura 3-15. Función get_features.**

### *3.2.2.6.2 Función transform_inputs*

Esta función se realiza mediante el método de concatenación que ofrece `numpy`. La función `np.concatenate()` permite unir o concatenar varios arrays a lo largo de un determinado eje.

En este caso (Figura 3-16) se tienen cinco arrays distintos, el formado por el análisis a las palabras y el análisis a los caracteres proporcionado por `TfidfVectorizer`, el devuelto por las características obtenidas del análisis con `get_features` y, por último, los correspondientes a la función de `CounterVectorizer`, tanto para palabras como caracteres. Estas estructuras usando axis= 1 se concatenarán a lo largo del eje horizontal, obteniendo así una matriz con todas las características extraídas del conjunto de datos, con las que ahora se podrá pasar a entrenar los distintos algoritmos. La función devuelve estos datos en formato de dataframe realizado por la librería `pandas`.

```
36  def transform_inputs(tweets,tfi_df,tfi_df2,CV,CV2):
37      features=tfi_df.fit_transform(tweets).toarray()
38      print('3')
39      features2=tfi_df2.fit_transform(tweets).toarray()
40      print('4')
41      oth_array=get_oth_features(tweets)
42      print('5')
43      bigram_vectorizer=CV.fit_transform(tweets).toarray()
44      bigram_vectorizer2=CV2.fit_transform(tweets).toarray()
45      M=np.concatenate([features, features2, oth_array],axis=1)
46      return pan.DataFrame(M)
```

**Figura 3-16. Función transform_inputs.**

La función de `get_features` necesita de un tratamiento previo a esta concatenación, por lo que se crea una función denominada `get_oth_features` (Figura 3-17) con la cual se obtendrá un array de arrays para cada tweet.

```
110  def get_oth_features(tweets):
111      feats=[]
112      for t in tweets:
113          feats.append(get_features(t))
114      return np.array(feats)
115
```

**Figura 3-17. Función get_oth_features.**

### *3.2.3 Entrenamiento de algoritmos*

Los algoritmos sobre los que se experimentan, para encontrar aquel que ofrezca un mejor resultado en la predicción requerida, son SVM, Multinomial NB, Gaussian NB y, por último, MLP y CNN, de los cuales se detallarán sus parámetros en este apartado. Los fundamentos de los algoritmos se pueden consultar en el apartado 2.1.2.

La librería de la que se obtienen todos los algoritmos en este caso es sklearn o Scikit-learn. Scikit-learn es una biblioteca de Python de código abierto para el aprendizaje automático. La biblioteca admite los últimos algoritmos, como KNN, XGBoost, SVM, etc. Está construido sobre `Numpy`. Scikit-learn se utiliza ampliamente en competiciones de Kaggle [46] y empresas de tecnología famosas. Scikit-Learn ayuda con el preprocesamiento, la reducción de dimensionalidad, clasificación, regresión, agrupación y selección de modelos.

El entrenamiento, de los algoritmos que se exponen a continuación, se realiza con el mismo conjunto de datos para comparar la eficiencia que se obtiene con cada uno de ellos.

Los pasos a seguir para entrenar a los algoritmos son los siguientes:

- Se aplica al dataset a tratar la transformación de los datos con la función de `transform_inputs`.
- Se divide el conjunto de datos en dos subconjuntos:
    - Conjunto de entrenamiento: Subconjunto para entrenar el modelo.
    - Conjunto de prueba: Subconjunto para probar el modelo entrenado.

Es necesario que el conjunto de datos que se utilice para el entrenamiento del modelo sea representativo, es decir, que reúna las características adecuadas para que el modelo genere los datos de prueba de manera correcta.

Esta división se realiza con la función automática de sklearn llamada `train_test_split`, eligiendo un subconjunto del 70% para entrenamiento y un subconjunto del 30% para las pruebas (Figura 3-18Figura 3-18).

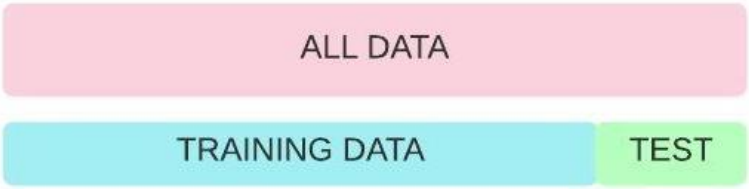


**Figura 3-18. Subconjuntos de datos.**

### *3.2.3.1 SVM*

Comenzando por el algoritmo de SVM, el parámetro kernel elegido es el predeterminado 'rbf', función de base radial, debido a que este es el kernel que puede crear los límites más complejos de los que dispone sklearn. En cambio, kernel lineal únicamente tiene un buen rendimiento en problemas muy sencillos, al igual que kernel polinomial divide en hiperplanos separadores, los cuales no son adecuados para la problemática de este trabajo (Figura 3-19).

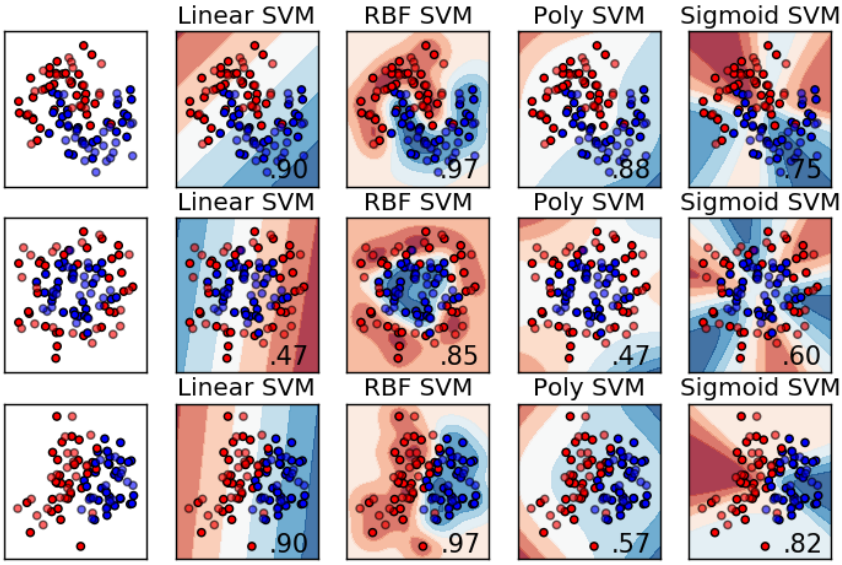


**Figura 3-19. Comparativas modelos kernel (extraído en [47]).**

### *3.2.3.2 Multinomial NB y GaussianNB*

En cuanto a los algoritmos derivados de la Teoría de Bayes, es decir, tanto MultinomialNB como GaussianNB son más sencillos de implementar, adoptando ambos los parámetros por defecto que proporciona la librería de sklearn, siendo por tanto la matriz que este algoritmo crea de predicción ajustada a los datos.

El algoritmo de Multinomial se considera adecuado para aplicar en características discretas, como por ejemplo podría ser el recuento de menciones (3.2.2.3); sin embargo, Gaussian se centra normalmente en predicciones relacionadas con características distribuidas, con lo cual se comprobará cuál de estos algoritmos se ajusta más a las necesidades de la cuestión de estudio.

### *3.2.3.3 MLP*

El primer algoritmo que se utilizará dentro del campo del aprendizaje profundo es Multi-layer Perceptron classifier. Los parámetros que se escogen en este caso serán 100 neuronas en la capa oculta enésima, las máximas iteraciones serán 9000, debido a que sino con la cantidad de datos en el corpus este modelo no llegaría a converger. En este caso se usa un escalador, el cual elimina la media y escala cada característica a la varianza de la unidad.

### *3.2.3.4 CNN*

El segundo algoritmo del aprendizaje profundo utilizado será CNN. El modelo secuencial implementado consta de 4 capas densamente conectadas disminuyendo la dimensionalidad de salida en cada capa, partiendo de 500 hasta llegar a la clasificación de clases, siendo esta la capa más cercana a la salida. Las activaciones iniciales de las capas serán en las tres primeras 'relu', lo que implica una activación de la unidad lineal rectificada, y siendo la capa más cercana activada mediante 'sigmoid', es decir, una función sigmoidea devolviendo esta un valor entre 0 y 1. Muestra de las capas implementadas se pueden observar en la Figura 3-20.

```
model = Sequential()
model.add(Dense(500, input_shape=(x_train.shape[1],)))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(250))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(50))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('sigmoid'))
```

**Figura 3-20. Capas modelo CNN.**

### *3.2.4 Métricas para resultados*

El último paso a realizar será evaluar el rendimiento de la clasificación. Para ello se usarán las métricas de precisión, recall, F1 y accuracy. Se define a continuación el significado de cada una de ellas:

- Precisión.
  Con esta métrica se mide la calidad del modelo en la tarea de la clasificación, respondiendo esta a la siguiente fórmula:
  $$precision = \frac{TP}{TP + FP}$$
  Correspondiendo por tanto la precisión a la división entre los verdaderos positivos entre la suma de todos los positivos.

- Exhaustividad (*recall*).
  Esta informará sobre la cantidad que el modelo es capaz de identificar. Correspondiéndose con:
  $$recall = \frac{TP}{TP + FN}$$
  En este caso se dividen los verdaderos positivos entre la clasificación total, es decir, tanto verdaderos positivos como falsos negativos.

- F1.
  Esta medida combina las medidas de precisión y de recall en un solo valor. Esta medida es la más útil, ya que permitirá comparar entre las soluciones de los distintos modelos. Responde a la media armónica entre la precisión y recall.
  $$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

- Exactitud (*accuracy*).
  Se obtiene el porcentaje de casos que el modelo ha devuelto correctamente. Este parámetro no debe llevar a error, ya que se puede considerar como un modelo malo mejor de lo que es. Se calcula a partir de la siguiente fórmula.

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

# 4 RESULTADOS / PRUEBA

Este capítulo está dedicado a la evaluación del sistema creado a partir de la aplicación de los algoritmos descritos en el apartado 3.2.3. Para lograr este objetivo, se hace necesario testearlo sobre un conjunto de datos sobre el que se extraerán las características descritas en el apartado 3.2.2. Se llegará así a evaluar el conjunto de algoritmos, aplicándose el conjunto de características descritas de forma incremental, y se indicará cuál de ellos ha proporcionado un rendimiento más elevado. Se comenzará por tanto describiendo el conjunto de datos sobre el que se ha realizado las pruebas.

## 4.1 El conjunto de datos

### *4.1.1 Conjunto de datos utilizados*

El corpus de datos utilizado para llevar a cabo el entrenamiento de los algoritmos fue cedido por la Universidad de Antwerp. Este conjunto de datos fue el que se utilizó en el trabajo presentado por los autores de [19] en el que ellos mismos presentaban un sistema de detección de odio. Este conjunto de tweets presenta una temática centrada en asuntos relacionados con el yihadismo y dispone de un total de 35.000 tweets clasificados como `safe` y alrededor de 49.000 etiquetados como `hate` (de odio).

El conjunto de datos proporcionado es un conjunto de tweets multilingüe, ya que contiene textos en inglés, árabe, francés, italiano, portugués, alemán, español, entre otros. Para llevar a cabo este trabajo, se ha decidido utilizar únicamente aquellos escritos en inglés, francés, español y árabe, por la facilidad de comprensión de los tres primeros idiomas y, por otro lado, debido a la temática considerada, un alto porcentaje de ellos estarán escritos en árabe. El objetivo será tratar de realizar una evaluación en función del idioma, motivo por el cual se hace necesario aplicar inicialmente la detección del idioma a cada tweet para extraer únicamente aquellos escritos en el idioma deseado. Esta separación se realiza ya que el sistema creado es dependiente del idioma, concretamente, en la realización del correspondiente preprocesado del texto.

La división del corpus se llevó a cabo a partir de una librería llamada `langdetect`.

#### *4.1.1.1 Langdetect*

`Langdetect` es un paquete Python simple el cual admite 55 idiomas diferentes para detectar (Figura 4-1), desarrollado por Michal Danilák.

```
af, ar, bg, bn, ca, cs, cy, da, de, el, en, es, et, fa, fi, fr, gu, he,
hi, hr, hu, id, it, ja, kn, ko, lt, lv, mk, ml, mr, ne, nl, no, pa, pl,
pt, ro, ru, sk, sl, so, sq, sv, sw, ta, te, th, tl, tr, uk, ur, vi, zh-cn, z
h-tw
```

**Figura 4-1. Código ISO idiomas langdetect (extraído de [48]).**

Esta librería permite determinar automáticamente el idioma en el que está redactado el mensaje, una vez introducido su contenido en la misma. Es importante aclarar que se trata de un algoritmo no determinista, lo que implica que al aplicarse en textos demasiado cortos o ambiguos podría obtenerse un resultado distinto cada vez que el mismo se ejecutase.

Por tanto, aplicando esta librería a los datos obtenidos de la Universidad de Antwerp, se han creado subconjuntos de tweets por cada uno de los idiomas. Cabe reseñar que, debido a que algunos de los tweets no presentaban la longitud suficiente o contenían caracteres no aceptados por la librería, no fue posible identificar su idioma, quedando clasificados en un archivo denominado “otros”.

Para mejorar el rendimiento de esta librería, se llevó a cabo una limpieza del texto, eliminándole tanto menciones como URLs. Ambas podían introducir ruido en el detector sin aportar información útil que sirviera para determinar el idioma.

El resultado son dos documentos por cada idioma separados en `hate/safe`. Cada fichero dispone de tres columnas: una con el identificador, otra con el tweet y la última de ellas con la clasificación de `hate/safe`. El conjunto obtenido puede verse en la Figura 4-2.

| Column1 | Column2 |
|---|---|
| 5,5545E+17 | " And We have put before them a barrier and behind them a barrier and covered them, so they do not see." - Surah Yasin:9 |
| 5,5544E+17 | The calm before the storm. |
| 5,5544E+17 | They said im a suspect in the beheadings and now they saying im the main suspect in the CENTCOM hack lol desperate journalism? Stupid kuffar |
| 5,5525E+17 | The hacking group CyberCaliphate is an independent hacking group that supports the state. It is not the official IS hacking group. . |
| 5,6234E+17 | As Long as Muslims are suffering there will be no peace in this planet.! spend all your budgets, all your skills, and all your experiences. |
| 5,6234E+17 | Every where Muslims are suffering their lands stolen, Kashimir,Palestine,Burma etc but everyday there is anti-terror conference. |
| 5,6234E+17 | #LRT a rare photo of 1982 Hama Massacre. |
| 5,6233E+17 | Sisi promised he will be destroying the entire city of Raffah -Sinai knowing this 50,k inhabitants live in Raffah. in other words its genocd |
| 5,6233E+17 | @MENCION with zero Media attention, then when His Army got killed as of a revenge for what they did. its every where BREAKING NEWS |
| 5,6233E+17 | Sisi and his Army destroyed an entire inhibited areas of Sinai, killed the old and the young.!! destroyed mosques,houses, with zero media |
| 5,6233E+17 | RT @MENCION: Reuters: Obama budget allocates $8.8 billion to fight Islamic State, help Iraq. |
| 5,6233E+17 | RT @MENCION: And then they say Women have no rights in Islam?! Psht. #HalalHumour http://t.co/zp33SWHUWq |
| 5,6232E+17 | RT @MENCION: I've always wanted to shout out Allahu Akbar on a plane but lol I doubt that will end well. |
| 5,6232E+17 | RT @MENCION: "If Europeans will accept to be tried in an African court, we will also accept to go to ICC" - Robert Mugabe. #Africa http://t.co/sBKwuDHzMw |
| 5,6232E+17 | RT @MENCION: Chinese? http://t.co/QuD3auXdo1 |
| 5,623E+17 | RT @MENCION: Did you know that the USA is the only country that sentences kids to life without parole? (h/t @MENCION) http://t.co/4G4KXjrDCp |
| 5,623E+17 | New Bay'ah from khorasan, Ameer of Tahreek e Taliban pakistan has given bayah to Islamicstate. http://t.co/8YfSsOZ5Nr |
| 5,6229E+17 | RT @MENCION: Bengali slave woman carries a British merchant on her back. 1903 West Bengal, India. http://t.co/pW3S7NQKIt |
| 5,6227E+17 | IS Sniper taking on peshmerga fighter https://t.co/6u7Xb0WaCB |
| 5,6226E+17 | Chad-Niger-Cameroon-Nigeria and some other west African nations sending troops to fight one single group. >>>B-H |

**Figura 4-2.Corpus hate tweets para el inglés[12].**

Posteriormente, para un manejo más sencillo de los archivos `csv` creados, se unen en un único archivo tanto aquellos de `hate` como de `safe`.

A la vista de los documentos obtenidos para cada idioma, tal y como se observa en la Figura 4-3, se decide trabajar con el idioma más numeroso, el inglés, pero también se decide realizar determinadas pruebas experimentales con el árabe.

[12] Se anonimizan las menciones contenidas en el corpus con el objetivo de mantener la confidencialidad de los datos, por ser estos datos reales y no ser el objeto del estudio presentado señalar a los usuarios.

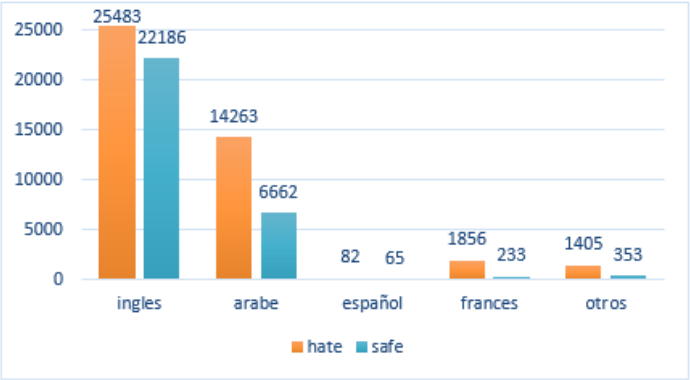


**Figura 4-3. Contenido corpus por idiomas.**

La primera evaluación de los algoritmos se llevará a cabo importando únicamente el corpus completo de tweets en inglés. Este corpus está formado por un conjunto de 25.483 tweet etiquetados como `hate`, representando un 56,46% del total; y 22186 tweets como `safe`, correspondiendo con un 46,54% del total, como se muestra en la Figura 4-3, por lo que podemos asumir que se encuentran balanceados.

No fue necesario un preprocesamiento en cuanto a la forma del documento se refiere, debido a que el archivo csv obtenido se encontraba bien diferenciado, separando las columnas por comas, es decir, el formato propio de este tipo de archivos, con el que era fácil trabajar a la hora de recorrer el documento.

## 4.2 Medición de resultados

En este apartado, como continuación al desarrollo del TFG se expresan los resultados obtenidos para las distintas pruebas realizadas con los distintos algoritmos. Inicialmente, se mostrarán las pruebas realizadas sobre la hipótesis de las características que inicialmente se consideran que serán las mejores. Posteriormente, estas pruebas se llevarán a cabo de forma incremental, comenzando únicamente con la inclusión de características superficiales, a las que se irán añadiendo más a través de una concatenación progresiva de características, tanto léxicas, como procedentes del análisis de sentimiento y de la polaridad de emojis.

### *4.2.1 Evaluación del sistema según algoritmo*

Considerando la combinación de las características de conteo de elementos de Twitter, la polaridad que proporciona `VaderSentiment` y la frecuencia inversa de unigramas, bigramas y bigramas de caracteres; los resultados que se obtienen tras la aplicación de las métricas de evaluación son las que se muestran en la Tabla 4-1.

| Algoritmos | Precisión | Recall | F1-score | Accuracy |
|---|---|---|---|---|
| SVM | 0,817 | 0,829 | 0,824 | 0,835 |
| MULTINOMIALNB | 0,752 | 0,779 | 0,765 | 0,777 |
| GAUSSIAN | 0,648 | 0,899 | 0,753 | 0,726 |
| MLP | 0,793 | 0,799 | 0,796 | 0,81 |
| CNN | 0,83 | 0,83 | 0,83 | 0,83 |

**Tabla 4-1. Evaluación simple algoritmos.**

En la Tabla 4-1, se comprueba como los algoritmos SVM y CNN, son los que mejor resultado obtienen en este contexto. La elección, en un primer momento, de este conjunto de características se hace teniendo en cuenta el aporte léxico, así como el análisis de sentimiento y objetos propios del lenguaje común de los posts, se prevé que tenga una precisión mayor.

Ahora bien, estos resultados fueron tomados sin tener en cuenta la realización de una validación cruzada. Para asegurar estos resultados, se realizará la comprobación verificando que no existe un sobreajuste en su entrenamiento. Se dividirá por tanto el conjunto de datos de entrenamiento en cinco grupos diferenciados. Cada uno de estos grupos servirá para realizar la validación del sistema, mientras que los otros cuatro servirán para entrenarlo. Al finalizar los cinco experimentos, se hallará la media, siendo los resultados finales obtenidos lo mostrados en la Tabla 4-2.

| **Algoritmos** | **Precisión** | **Recall** | **F1-score** | **Accuracy** |
|---|---|---|---|---|
| SVM | 0,809 | 0,821 | 0,815 | 0,823 |
| MULTINOMIALNB | 0,749 | 0,76 | 0,754 | 0,77 |
| GAUSSIAN | 0,644 | 0,9 | 0,75 | 0,722 |
| MLP | 0,787 | 0,783 | 0,785 | 0,8 |
| CNN | 0,83 | 0,828 | 0,828 | 0,83 |

**Tabla 4-2. Evaluación algoritmos cross.**

Teniendo en cuenta la Tabla 4-2, para realizar las experiencias y comprobar cuáles son las características que elevan la precisión del modelo se llevarán a cabo con el mejor de los algoritmos de *Machine Learning*, es decir, con SVM; y con el algoritmo de *Deep learning* denominado MLP, escogido en lugar de CNN, debido a la complejidad de implementación de este último para realizar pruebas, teniendo en cuenta además la escasez de tiempo para realizar todas las pruebas requeridas con este algoritmo (algunas de las pruebas realizadas pueden llegar a tardar días).

### *4.2.2 Evaluación incremental de las características*

En estas pruebas se aplican de forma incremental combinaciones de características. Se presentarán en la Tabla 4-3 tres bloques de pruebas para los dos algoritmos escogidos. El primer bloque, estará formado por las características denominadas superficiales, añadiéndole en los últimos experimentos la polaridad de sentimiento y el conteo de elementos propios del lenguaje de Twitter. El segundo de los bloques implementa tanto `CounterVectorizer` como la unión con las características superficiales utilizadas en el bloque anterior. En el último de los bloques, se utilizan las características léxicas aportadas por `TfidfVectorizer` añadidas, en este caso, únicamente a la polaridad de sentimiento y conteo de menciones, hashtags y emojis.

Es importante resaltar que el aumento incremental de las características no es riguroso entre bloques, eliminando de la tabla aquellas pruebas que descienden los resultados en pruebas anteriores, por lo que no se tienen en cuenta en pasos posteriores. Además, se ve la necesidad de citar que las pruebas que se muestran tienen un elevado coste computacional y temporal, realizando por tanto únicamente las pruebas que, a la vista de los resultados anteriores puedan mejorarlos. Las características utilizadas se encuentran detalladas en el apartado 3.2.2.

| Algoritmo | | Características | Precisión | Recall | F1-score | Accuracy |
|---|---|---|---|---|---|---|
| SVM | (1) | Num_terms | 0,519 | 0,199 | 0,288 | 0,542 |
| | (2) | (1)+num_chars_total | 0,52 | 0,199 | 0,288 | 0,542 |
| | (3) | (2) + num_words | 0,592 | **0,404** | **0,480** | **0,593** |
| | (4) | (3) +num_chars | **0,593** | 0,398 | 0,476 | 0,592 |
| | (5) | (4) + SentimentVader | 0,592 | 0,4 | 0,477 | 0,592 |
| | (6) | (5) + twitter_objs | 0,591 | 0,402 | 0,479 | 0,592 |
| | (7) | CounterVectorize (unigramas, bigramas) | 0,795 | 0,76 | 0,777 | 0,797 |
| | (8) | (4) + (7) | 0,694 | 0,42 | 0,523 | 0,644 |
| | (9) | CounterVectorize (2chgrams, 3chgrams) | 0,812 | 0,815 | 0,813 | 0,826 |
| | (10) | (4) + (9) | 0,786 | 0,71 | 0,746 | 0,775 |
| | (11) | (7) + (9) | **0,814** | **0,816** | **0,815** | **0,828** |
| | (12) | (4) + (11) | 0,742 | 0,75 | 0,746 | 0,762 |
| | (13) | SentimentVader+twitter_objs+tfidf(words) | 0,7877 | 0,769 | 0,778 | 0,796 |
| | (14) | SentimentVader+twitter_objs+tfidf(char) | 0,782 | 0,788 | 0,785 | 0,799 |
| | (15) | SentimenTextBlob+twitter_objs+tfidf(words) | 0,7524 | 0,731 | 0,742 | 0,763 |
| | (16) | SentimentTextBlob+twitter_objs+tfidf(char) | 0,782 | 0,789 | 0,785 | 0,799 |
| | (17) | (13)+ tfidf(chars) | 0,809 | 0,812 | 0,810 | 0,823 |
| | (18) | (17) +emojis | **0,809** | **0,814** | **0,811** | **0,824** |
| MLP | (1b) | Num_terms | 0,522 | 0,296 | 0,378 | 0,545 |
| | (2b) | (1b)+num_chars_total | 0,517 | 0,278 | 0,362 | 0,54 |
| | (3b) | (2b) + num_words | 0,625 | 0,381 | 0,473 | 0,60 |
| | (4b) | (3b) +num_chars | **0,633** | 0,372 | 0,469 | 0,605 |
| | (5b) | (4b) + SentimentVader | 0,596 | 0,46 | 0,519 | 0,604 |
| | (6b) | (5b) + twitter_objs | 0,601 | **0,491** | **0,540** | **0,614** |
| | (7b) | CounterVectorize (unigramas, bigramas) | 0,758 | 0,743 | 0,750 | 0,7696 |
| | (8b) | (4b)+(7b) | 0,756 | 0,754 | 0,755 | 0,773 |
| | (9b) | CounterVectorize (2chgrams, 3chgrams) | **0,83** | **0,83** | **0,830** | **0,843** |
| | (10b) | (4b)+(9b) | 0,816 | 0,805 | 0,81 | 0,824 |
| | (11b) | (7b)+(9b) | 0,812 | 0,818 | 0,815 | 0,827 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | (12b) | (4b)+(11b) | 0,813 | 0,819 | 0,816 | 0,827 |
| | (13b) | SentimentVader+twitter_objs+tfidf(words) | 0,726 | 0,71 | 0,718 | 0,74 |
| | (14b) | SentimentVader+twitter_objs+tfidf(char) | 0,766 | 0,769 | 0,767 | 0,783 |
| | (15b) | SentimenTextBlob+twitter_objs+tfidf(words) | 0,73 | 0,711 | 0,720 | 0,74 |
| | (16b) | SentimentTextBlob+twitter_objs+tfidf(char) | 0,764 | 0,759 | 0,761 | 0,779 |
| | (17b) | (15b)+tfidf(char) | 0,778 | 0,781 | 0,779 | 0,795 |
| | (18b) | (17b)+emojis | **0,786** | **0,777** | **0,781** | **0,798** |

**Tabla 4-3. Pruebas SVM y MLP.**

En el tercer bloque de características, tanto para SVM como MLP, no se muestran las pruebas correspondientes al analizador de sentimiento únicamente, ni la compuesta del analizador y el conteo de elementos de twitter, por no ofrecer un resultado que mejorara los anteriores. Además, se realizan pruebas con los dos analizadores de sentimiento, utilizándose en SVM el proporcionado por `VaderSentiment`, ya que ofrece una medida F1 mayor, y en el caso de MLP, se utiliza `TextBlob` por la misma razón citada anteriormente. No se muestra una prueba conjunta, a mayores, con las combinaciones de los mayores resultados de cada bloque por conllevar un alto coste temporal.

Según los resultados obtenidos con SVM, se puede comprobar como la combinación de las características numéricas con las de polaridad no aportan demasiada información, obteniendo como medida más alta de F1 un 48%, lo cual no permitirá al sistema discernir el contenido de odio. Las pruebas con `CounterVectorizer` refuerzan aún más la idea de que incluso eliminando la frecuencia inversa que obteníamos con `TfidfVectorizer` obtenemos mejores resultados, llegando a una medida de F1 de un 81,5%. En cuanto a la unión con las características léxicas aportadas por `TfidfVectorizer` provoca un gran aumento en la medida F1, consiguiendo hasta un 81,1%, lo cual muestra la importancia de las características léxicas a la hora de realizar la detección que se desea.

Pasando al algoritmo de MLP, se obtienen unos resultados parecidos a los del algoritmo anterior, aunque en este caso, se tiene una medida F1 de un 54% para las características numéricas, un 83% con `CounterVectorizer`, siendo este el valor más alto de las pruebas llevadas a cabo, y, un 78,1% para la concatenación con las características léxicas de `TfidfVectorizer`.

A la vista de estos resultados, se lleva a cabo una comparativa entre los resultados obtenidos por la Universidad de Antwerp, utilizando ellos, en su caso, conjunto de trigramas de caracteres (desarrollado en el apartado 2.2.1.1).

El resultado de la Universidad para los textos en inglés fue de una medida de F1 de un 79%, el cual queda por debajo del mejor de los resultados anteriormente expuestos, siendo este un valor máximo de 83% de medida F1 para el algoritmo de MLP utilizando tanto bigramas como trigramas de caracteres.

Por último, es necesario reseñar que, la única prueba realizada con CNN ofrecía una medida F1 de 82,8%, valor igualado a la mejor de las pruebas realizadas con los algoritmos de SVM y MLP. Se supera así también el valor del resultado obtenido por la Universidad de Antwerp.

### *4.2.3 Matrices de confusión*

Para evaluar las gráficas, es necesario recordar que los clasificadores `hate/safe`, tomaron los valores de 0 y 1, por lo que se comprobará con estas matrices del conjunto de test cuántos de estos tweets fueron clasificados correctamente y cuales dieron falsos positivos y falsos negativos.

Se muestran así las matrices correspondientes a la prueba inicial teniendo en cuenta todos los algoritmos, y, a continuación, aquellas matrices para el mejor resultado obtenido en las pruebas con validación cruzada (Figura 4-4, Figura 4-5, Figura 4-6, Figura 4-7, Figura 4-8, Figura 4-9, Figura 4-10).

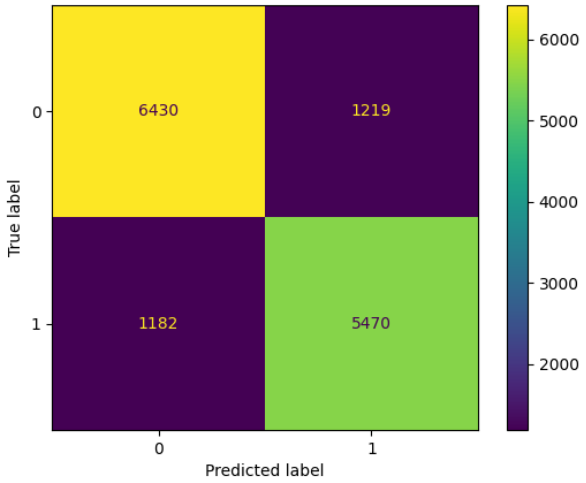


**Figura 4-4. Matriz confusión SVM.**

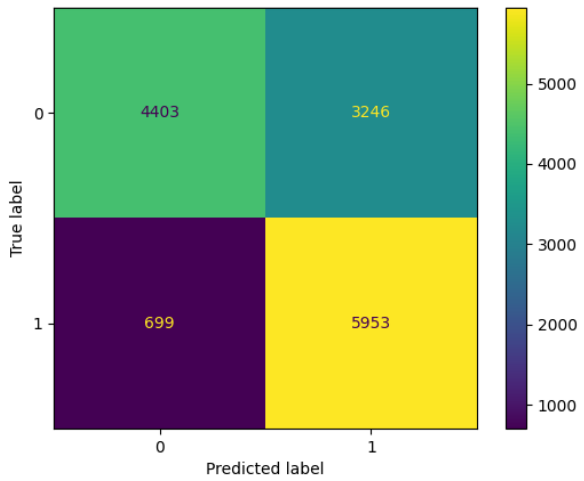


**Figura 4-5. Matriz de confusión GaussianNB.**

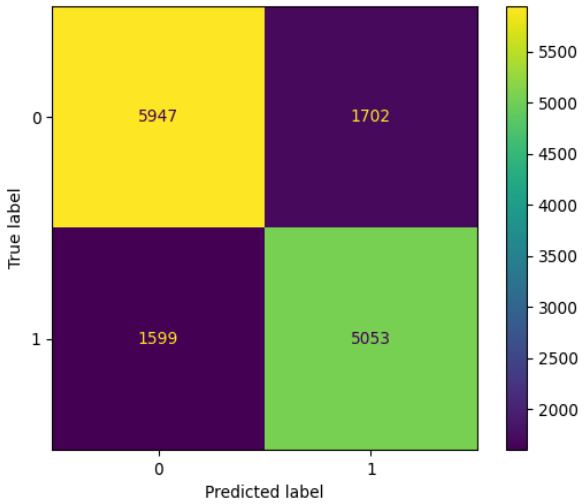


**Figura 4-6. Matriz confusión MultinomialNB.**

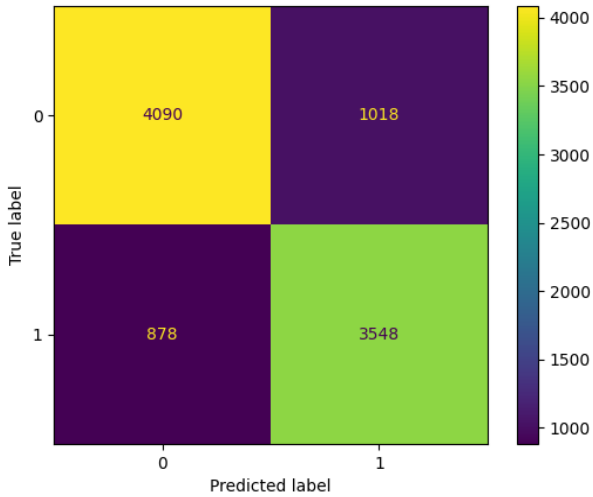


**Figura 4-7. Matriz confusión MLP.**

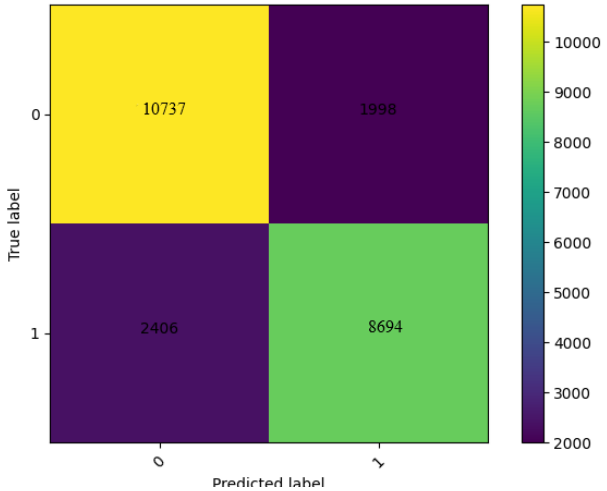


**Figura 4-8. Matriz confusión CNN.**

Estas matrices de confusión muestran de una manera más visual, cuanto de precisos son los modelos presentados, teniendo estos un mejor resultado cuantos mayores sean los valores de la diagonal principal en comparación con los que se encuentran fuera de esta.

Se puede observar, por tanto, como los algoritmos de SVM y MLP fueron escogidos para la realización de la extracción de características incrementales, obteniendo así el modelo con mayor precisión, al tener estos su diagonal principal más diferenciada de los datos alrededor de esta.

Se llega por tanto a las matrices de confusión de los algoritmos de SVM, como MLP, con las características que mejor resultado ofrecieron en las pruebas experimentales con características incrementales, siendo estas las obtenidas por `CounterVectorizer`, teniendo en cuenta tanto unigramas como bigramas de palabras, así como bigramas y trigramas de caracteres en el caso de SVM, obteniendo una medida F1 de 81,5% (Figura 4-9); y por otro lado, realizando un análisis de los conjuntos de bigramas y trigramas de caracteres, obteniendo una medida de F1 83% para MLP (Figura 4-10).

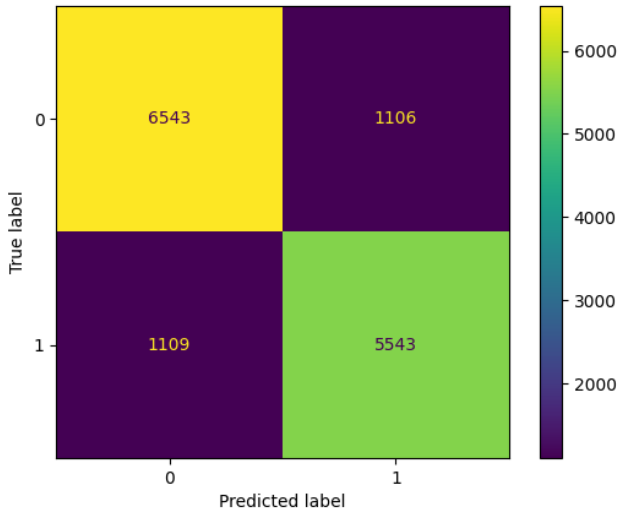


**Figura 4-9. Matriz Confusión SVM CounterVectorizer.**

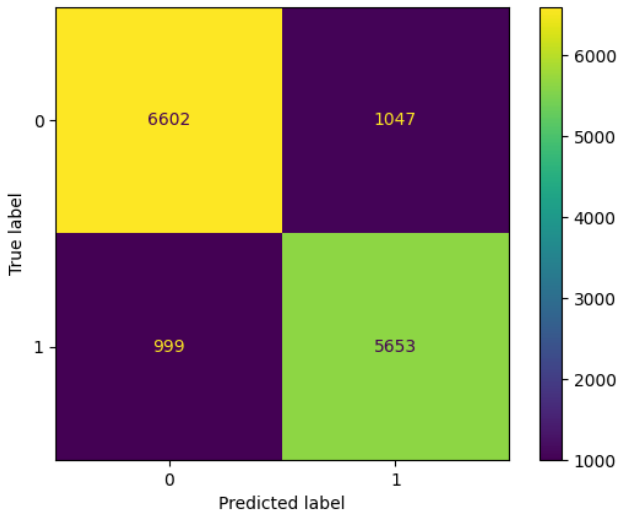


**Figura 4-10. Matriz Confusión MLP CounterVectorizer.**

### *4.2.4 Pruebas experimentales con árabe*

Para llevar a cabo estas pruebas fue necesario realizar un balanceo previo del corpus de tweets, debido a que la cantidad de tweets clasificados como `hate` era más elevado que aquellos clasificados como `safe`.

Quedando con un corpus formado por los datos expuestos en la Figura 4-11:

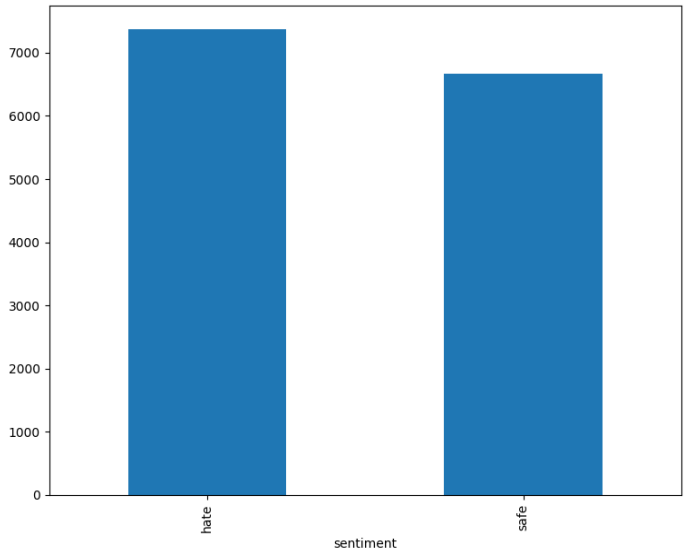


**Figura 4-11. Corpus tweet árabe.**

En este caso no se implementó el detector de sentimientos, debido a que este no contiene léxico para detectar la polaridad en este idioma. Se encuentran implementadas en el código las demás características utilizadas en el corpus de inglés igualmente, adaptadas para el idioma arábico, a excepción de la característica mencionada anteriormente.

Tomando aquellas características utilizadas en las pruebas con mejores resultados para el inglés, se realiza una prueba con cada algoritmo, implementando una validación cruzada de cinco estratificaciones, es decir, `CounterVectorizer` teniendo en cuenta bigramas y trigramas de caracteres.

| **Algoritmos** | **Características** | **Precisión** | **Recall** | **F1-score** | **Accuracy** |
|---|---|---|---|---|---|
| SVM | CounterVectorize(2chgrams, 3chgrams) | **0,868** | 0,863 | 0,865 | 0,873 |
| GAUSSIANNB | CounterVectorize(2chgrams, 3chgrams) | 0,78 | **0,88** | 0,827 | 0,826 |
| MULTINOMIALNB | CounterVectorize(2chgrams, 3chgrams) | 0,823 | 0,80 | 0,811 | 0,825 |
| MLP | CounterVectorize(2chgrams, 3chgrams) | 0,854 | 0,869 | 0,861 | 0,867 |
| CNN | CounterVectorize(2chgrams, 3chgrams) | **0,88** | 0,878 | **0,878** | **0,88** |

**Tabla 4-4. Evaluación algoritmos árabe.**

Se muestran, a continuación, las matrices de confusión para la evaluación realizada en la Tabla 4-4. Las matrices se corresponden con las figuras que siguen: Figura 4-12, Figura 4-13, Figura 4-14, Figura 4-15 y Figura 4-16.

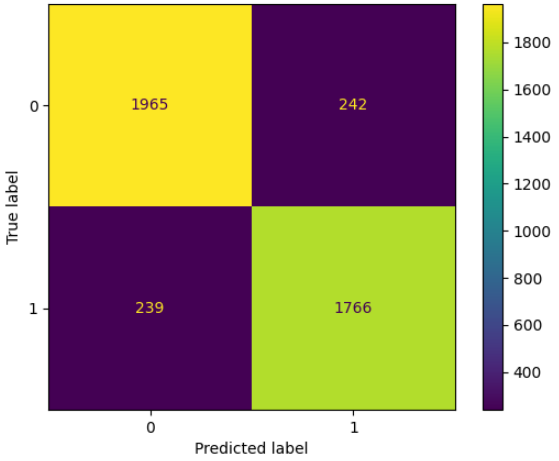


**Figura 4-12. Matriz confusión SVM árabe.**

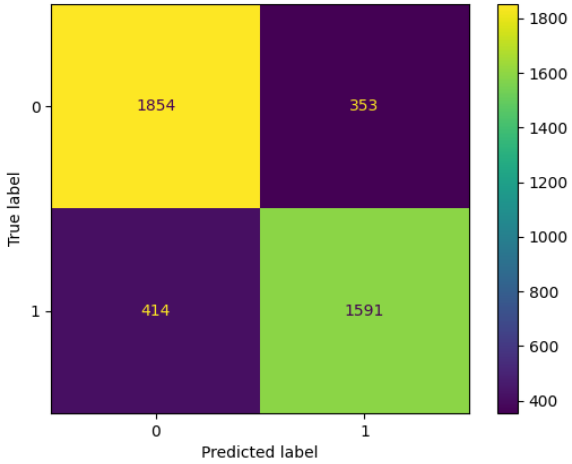


**Figura 4-13. Matriz MultinomialNB árabe.**

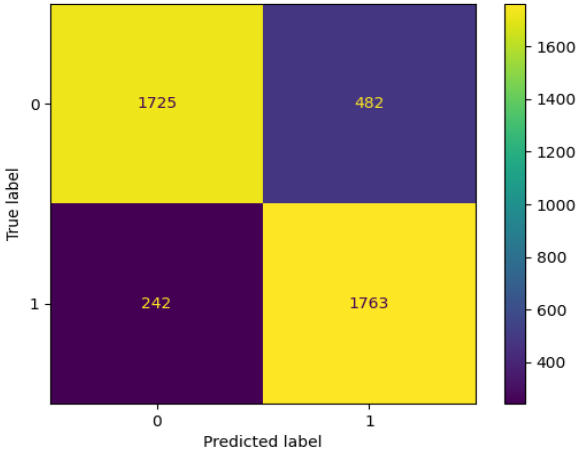


**Figura 4-14. Matriz confusión GaussianNB árabe.**

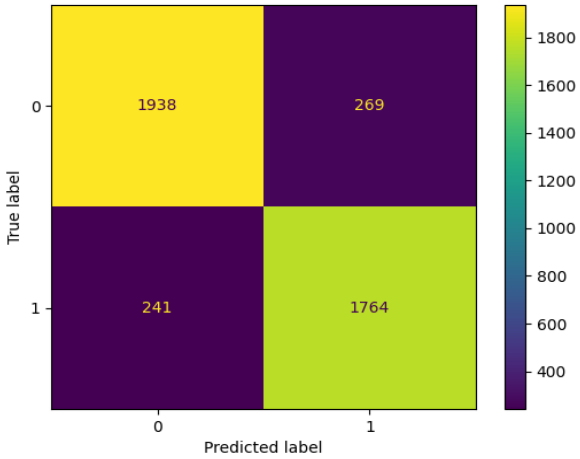


**Figura 4-15. Matriz confusión MLP árabe.**

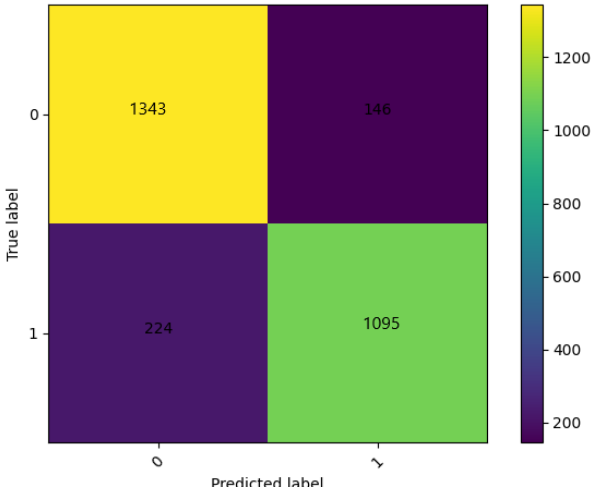


**Figura 4-16. Matriz confusión CNN árabe.**

Se comprueba por tanto cómo, aunque obteniendo resultados parecidos, los algoritmos más precisos coinciden al igual que en inglés con SVM, en el caso de *Machine Learning* y CNN, en el caso de *Deep Learning*, obteniendo unos resultados de medida F1 de 86,5% y 87,8% respectivamente.

Al igual que en el caso del corpus en inglés, comparándolo con los resultados de Antwerp, en este caso su medida F1 era de un 84%, obteniendo en el mejor de nuestros casos un 87,8%, es decir, un 3,8% superior.

Teniendo en cuenta estos resultados, se observa cómo se obtienen mejores medidas F1 en comparación con el corpus de inglés. Se muestra a continuación en la Tabla 4-5 la traducción de tres tweets de ejemplo, mostrándose así el tipo de contenido que se puede encontrar en estos datos.

| |
|---|
| *"RT @ xxxxx: Oh Dios, te pido que lo captures # el Estado Islámico para quemarlo como quemó los corazones de las madres en # Raqqa y librar a # Jordan de él, te estamos esperando Queenie http://t.co / uCBebP4FXZ* |
| *RT @ xxxxx: Una de las escenas más bellas*<br>*Moaz kebab está ardiendo*<br>*# ISIS_Burning_Jordanian Pilot*<br>*#Jordán*<br>*https://t.co/2rxjQ7Zc65* |
| *RT @ xxxxx: "@ xxxxx: youtube Estado Islámico # Al-Raqqa Province # Musulmanes celebran la quema del piloto jordano https://t.co/3U0abXdpOE "* |

**Tabla 4-5: Ejemplos tweets árabes traducidos**

Las causas de este aumento en los resultados pueden deberse a varios hechos, como pueden ser la temática del corpus con el que trabajamos, o el idioma. El input del idioma es importante reseñarlo, ya que, al no ser un idioma muy utilizado en las redes sociales, puede desembocar en unos tweets con contenido más agresivo que en otros casos, por lo que, en contraposición, es más fácil distinguir entre aquellos tweets que contienen un mensaje de odio a los que no.

# 5 CONCLUSIONES Y LÍNEAS FUTURAS

## 5.1 Conclusiones

Concluyendo la línea de resultados, se llega a una serie de ideas principales. Una de ellas basada en las características que tienen más influencia a la hora de detectar el discurso de odio en las redes sociales, siendo estas características las denominadas léxicas. En contraposición, aquellas características consideradas como superficiales, es decir, únicamente el conteo numérico de caracteres, palabras, longitud del tweet; no aportaban precisión al sistema, llegando incluso a disminuir sus resultados en algunos casos.

Por otro lado, siguiendo en la línea relacionada con la extracción de determinadas características, se encuentra la necesidad de aplicar un analizador de sentimiento. A la vista de los resultados, se puede observar que el analizador de sentimiento `VaderSentiment` proporciona menos exactitud que `TextBlob` para el algoritmo MLP. Sin embargo, en el caso del algoritmo SVM, `VaderSentiment` es el que proporciona un mejor rendimiento. A pesar de existir esta diferencia, la variación de rendimiento es mínima, pero cabe reseñar que con `TextBlob`, se permite la inclusión de, además de la polaridad de la oración, su puntuación de sarcasmo.

A la vista del estado del arte, se puede observar cómo se mejoran algunos de los resultados que los distintos autores obtienen en los trabajos presentados. Comparando con la Universidad de Antwerp, ya que el conjunto de datos del que se dispone es el mismo, ellos obtienen una medida F1 de 79% en el caso de inglés y un 84% para el árabe. Por lo tanto, el sistema creado mejora en ambos casos entorno a un 4% los resultados de los investigadores.

Por último, cabe resaltar que los objetivos del trabajo de fin de grado se completaron en su totalidad.

## 5.2 Líneas futuras

Tras la realización del detector de odio, se aprecia la pérdida de información necesaria al eliminar en el preprocesado las url que se encuentran añadidas a los tweets. Estas url´s suelen estar eliminadas de la web, ya que, es probable que contuvieran contenido sensible.

Es por ello, que se hace necesario seguir investigando en implementar una búsqueda de que contenido se presenta en esas url´s para llevar así a cabo una clasificación de los tweets con contenido de odio más precisa que la que resulta en el presente trabajo. Además, se puede añadir el análisis de memes, los cuales se encuentran en pleno auge en la sociedad actual.

Por otro lado, se propone la mejora del detector en el idioma de árabe, debido a que mucho contenido en la red relacionado con el yihadismo se encuentra escrito en este idioma, por lo que se

podría añadir, al igual que en inglés un detector de sentimiento. Relacionado con ello, se propone la aplicación del modelo de redes convolucionales una vez se implemente el detector de sentimiento para poder realizar así una comparativa con los demás modelos implementados.

Por último, algunos de los emojis que los tweets contenían no se encontraban codificados en UNICODE, por lo que no se pudieron recuperar al realizar el análisis de los mismos. Se propone así realizar una búsqueda más profunda de los emojis no codificados en el lenguaje estándar para poder tenerlos en cuenta en el análisis del corpus.

# 6 BIBLIOGRAFÍA


[1] I. C. Sánchez, «Francia lanza una ofensiva contra la incitación al odio en las redes sociales,» *elPeriódico,* 19 Octubre 2020.

[2] N. ONU, «nws.un.org,» Paz y seguridad, 19 Noviembre 2020. [En línea]. Available: https://news.un.org/es/story/2020/11/1484342.

[3] J. T.Nockleby, «Hate Speech,» de *Encyclopedia of the American Constitution*, New York, Macmillian Publishers, 2000, pp. 1277-1279.

[4] C. o. M. t. m. S. R. N. R. 20, 1997. [En línea]. Available: https://rm.coe.int/CoERMPublicCommonSearchServices/.

[5] Calgacus, «http://www.redbubble.com/people/calgacu,» [En línea].

[6] C. EADIC. [En línea]. Available: https://www.eadic.com/inteligencia-artificial-vs-aprendizaje-automatico-vs-deep-learning-vs-ciencia-de-datos/.

[7] «Netflix,» [En línea]. Available: https://www.netflix.com/es/.

[8] «Amazon Alexa,» [En línea]. Available: https://developer.amazon.com/es-ES/alexa.

[9] BBVA, «Machine Learning:¿qué es y cómo funciona?,» [En línea]. Available: https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/.

[10] Scikit-Learn, 2017. [En línea]. Available: https://scikit-learn.org/stable/modules/svm.html.

[11] A. Navlani, 2019. [En línea]. Available: https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python.

[12] Scikit-Learn, 2017. [En línea]. Available: https://scikitlearn.org/stable/modules/naive_bayes.html.

[13] Scikit-Learn. [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html?highlight=multinomial%20nb#sklearn.naive_bayes.MultinomialNB.

[14] Scikit-Learn, 2017. [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html?highlight=gaussia

n%20nb#sklearn.naive_bayes.GaussianNB.

[15] «scikit-learn.org,» [En línea]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes.

[16] «HZ hard zone,» [En línea]. Available: https://hardzone.es/tutoriales/rendimiento/diferencias-ia-deep-machine-learning/.

[17] Scikit-Learn. [En línea]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[18] J. Martinez. [En línea]. Available: https://datasmarts.net/es/redes-neuronales-convolucionales-en-profundidad/.

[19] G. D. P. a. P. V. O. Tom De Smedt, «Automatic detection of online jihadist hate speech.,» Computational linguistics and psycholinguistics , University of Antwerp, 2018.

[20] D. H. Zeerak Waseem, «Hateful Symbols or Hateful People? Predictive features for Hate Speech detection on Twitter».

[21] W. W, L. Chen y K. a. S. Thirunarayan, «Cursing english on twitter,» A.P, 2014.

[22] I. a. W. Y. Kwok, «Locate the hate: Detecting tweets against blacks.,» AAAI, 2013.

[23] «https://www.researchgate.net/figure/Bag-of-Words-example_fig1_339076675,» [En línea].

[24] D. B. I. W. Thomas Davidson, «Racial Bias in Hate Speech and Abusive Language Detection Datasets».

[25] D. W. M. M. I. W. Thomas Davidson, «Automated hate speech detection and the problem of offensive language,» ICWSM '17, Montereal, Canada, 2017.

[26] a. H. J. Warner W, «Detectinghate speech on the world wide web,» SM 19-26, 2012.

[27] D. Yarowsky, «DECISION LISTS FOR LEXICAL AMBIGUITY RESOLUTION,» Department of Computer and Information Science, University of Pennsylvania, Philadelphia.

[28] TIOBE, «índice|TIOBE,» [En línea]. Available: https://www.tiobe.com/tiobe-index/.

[29] P. p. p. NumFOCUS, «pandas,» [En línea]. Available: https://pandas.pydata.org/.

[30] F. a. V. G. a. G. A. a. M. V. Pedregosa, «Scikit-learn: Machine Learning in {P}ython,» *Journal of Machine Learning Research,* vol. 12, nº 2011, pp. 2825--2830.

[31] S. E. L. y. E. K. Bird, Procesamiento del lenguaje natural con Python, O'Reilly Media Inc, 2009.

[32] T. Oliphant, «Numpy,» [En línea]. Available: https://numpy.org/.

[33] J. Hunter, NumFOCUS, [En línea]. Available: https://matplotlib.org.

[34] F. Chollet, «Keras,» [En línea]. Available: https://keras.io.

[35] J. D. R. Monga, Google Brain, [En línea]. Available: https://www.tensorflow.org/.

[36] C. y. G. E. Hutto, VADER: Un modelo parsimonioso basado en reglas para el análisis de sentimientos de texto de redes sociales. Octava Conferencia Internacional sobre Weblogs y Redes Sociales (ICWSM-14), Ann Arbor, MI, 2014.

[37] Kyokomi, «pypi,» [En línea]. Available: https://pypi.org/project/emoji/.

[38] P. Foundation. [En línea]. Available: https://docs.python.org/3/library/unicodedata.html.

[39] «TextBlob,» [En línea]. Available: https://textblob.readthedocs.io/en/dev/py-modindex.html.

[40] «Visual Code Studio,» [En línea]. Available: https://code.visualstudio.com/.

[41] P. Pandey, «Simplifying Sentiment Analysis using VADER in Python (on Social Media Text),» [En línea]. Available: https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f.

[42] Anurag, «newgenapps,» [En línea]. Available: https://www.newgenapps.com/blog/the-secret-way-of-measuring-customer-emotions-social-media-sentiment-analysis/.

[43] O´REILLY. [En línea]. Available: https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html.

[44] J. D. U. Jure Leskovec, «Data Mining,» de *Mining of Massive Datasets*, California, Cambridge University Press, 2014, pp. 1-18.

[45] p. |. a. SEO. [En línea]. Available: https://puntorojo.agency/tf-idf-term-frequency-inverse-document-frequency/.

[46] «kaggle,» [En línea]. Available: https://www.kaggle.com/.

[47] A. Bilogur, «kaggle.com,» [En línea]. Available: https://www.kaggle.com/residentmario/kernels-and-support-vector-machine-regularization.

[48] «PyPI,» [En línea]. Available: pypi.org/project/langdetect/.

[49] R. {. v. R. e. {. r. {. v. e. y. P. Sojka, «Marco de software para el modelado de temas con grandes corporaciones,» de *Actas del taller de LREC 2010 sobre nuevos*, ELRA, 2010, pp. 45--50.

[50] J. J.-M. S. G.-M. E. C.-M. F. J. G.-C. Milagros Fernández-Gavilanes, «Creating emoji lexica from unsupervised sentiment analysis of their descriptions,» de *Expert Systems with Applications*, ELSEVIER, 2018, pp. 74-91.

# ANEXO I: CÓDIGO

## 6.1 Separador idiomas

```
import re
from langdetect import detect
import pandas as pd
import csv


filename="IS-hate.csv"
filename2="IS-safe.csv"

## Funcion para detectar caracteres raros
def v(s):
    s = s.lower()
    s = re.sub(r'#', '', s)# borra hashtags
    s = re.sub(r'@[\w_]+', '', s)            # b usernames
    s = re.sub(r'https?://[^\b]+', '', s)  # strip URLs
    return s

##funcion para crear archivos segun el idioma detectado
def messages_class(language):
    hate="IS-hate-{}.csv".format(language)
    idioma=open(hate, 'a', newline="")
    return idioma
def messages_class_safe(language):
    safe="IS-safe-{}.csv".format(language)
    idioma=open(safe,'a', newline="")
    return idioma

## Funcion prinicipal donde se ejecuta todo el programa
def main():
    hate_data=pd.read_csv(filename)
    safe_data=pd.read_csv(filename2)
    df=pd.DataFrame.from_dict(hate_data)
    df2=pd.DataFrame.from_dict(safe_data)

    for i,text in enumerate(df.iloc[:,2]):
        try:
            language=detect(v(text))
            print("esta escrito en :",language)
            if language=="en":
                file=df.iloc[i,[0,2]]
                writer=csv.writer(messages_class(language))
                writer.writerow(file)
            if language=="ar":
                file=df.iloc[i,[0,2]]
                writer=csv.writer(messages_class(language))
                writer.writerow(file)
            if language=="es":
                file=df.iloc[i,[0,2]]
                writer=csv.writer(messages_class(language))
```

```
50                  writer.writerow(file)
51              if language=="fr":
52                  file=df.iloc[i,[0,2]]
53                  writer=csv.writer(messages_class(language))
54                  writer.writerow(file)
55          except:
56                  print(language, "This row throws and error", i)
57                  writer=csv.writer(messages_class("other"))
58                  writer.writerow(df.iloc[i])
59                  print(df.iloc[i])
60
61      for i,text in enumerate(df2.iloc[:,2]):
62          try:
63              language=detect(v(text))
64              print("esta escrito en :",language)
65              if language=="en":
66                  file=df2.iloc[i,[0,2]]
67                  writer=csv.writer(messages_class_safe(language))
68                  writer.writerow(file)
69              if language=="ar":
70                  file=df2.iloc[i,[0,2]]
71                  writer=csv.writer(messages_class_safe(language))
72                  writer.writerow(file)
73              if language=="es":
74                  file=df2.iloc[i,[0,2]]
75                  writer=csv.writer(messages_class_safe(language))
76                  writer.writerow(file)
77              if language=="fr":
78                  file=df2.iloc[i,[0,2]]
79                  writer=csv.writer(messages_class_safe(language))
80                  writer.writerow(file)
81          except:
82                  print(language, "This row throws and error", i)
83                  writer=csv.writer(messages_class_safe("other"))
84                  writer.writerow(df2.iloc[i])
85                  print(df2.iloc[i])
86
87  if __name__ == "__main__":
88      main()
89
```

**Figura A1-1 Separador idiomas**

## 6.2 Algoritmos

```
#SVM
def SVM_algorimt(X, y):

    X_train, X_test, y_train, y_test = train_test_split (X,y,test_size=0.3,random_state=45)
    kf=KFold(n_splits=5)

    support_vector = svm.SVC(C=10, kernel='rbf', cache_size=200, class_weight=None,coef0=0.0,
    decision_function_shape='ovr', degree=3,gamma='scale',max_iter=-1,probability=False,
    random_state=None, shrinking=True,tol=0.001, verbose=False)
    support_vector.fit(X_train, y_train)

    #score= support_vector.score(X_train, y_train)
    #print('Metrica del modelo',score)
    scoring={'accuracy':'accuracy','recall':'recall','precision':'precision'}
    scores=cross_validate(support_vector,X_train,y_train,cv=kf, scoring=scoring)

    print('Metricas cross_validation',scores)

    #y_pred = support_vector.predict(X_test)
    plot_confusion_matrix(support_vector,X_test,y_test)
    print(plt.show())
    #print('Accuracy score_svm: ', format(accuracy_score(y_test, y_pred)))
    #print('precision score_svm: ', format(precision_score(y_test, y_pred)))
    #print('recall score_svm: ', format(recall_score(y_test, y_pred)))
    #print('f1 score_svm: ', format(f1_score(y_test, y_pred)))

#MULTINIMIALNB
def Multinomialnb(X, y):

    X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3, random_state = 45)
    kf=KFold(n_splits=5)
    clf = MultinomialNB().fit(X_train, y_train)

    #score= clf.score(X_train, y_train)
    #print('Metrica del modelo',score)
    scoring={'accuracy':'accuracy','recall':'recall','precision':'precision'}
    scores=cross_validate(clf,X_train,y_train,cv=kf, scoring=scoring)

    print('Metricas cross_validation',scores)

    #y_pred = clf.predict(X_test)
    plot_confusion_matrix(clf,X_test,y_test)
    print(plt.show())
    #print('Accuracy score_multi: ', format(accuracy_score(y_test, y_pred)))
    #print('precision score_multi: ', format(precision_score(y_test, y_pred)))
    #print('recall score_multi: ', format(recall_score(y_test, y_pred)))
    #print('f1 score_multi: ', format(f1_score(y_test, y_pred)))
```

**Figura A1-2. Implementación SVM y MultinomialNB.**

```python
def Gaussiannb(X,y):
    X_train, X_test, y_train, y_test = train_test_split (X,y,test_size=0.3,random_state=45)
    gnb = GaussianNB()
    kf=KFold(n_splits=5)
    gnb.fit(X_train, y_train)

    #score= gnb.score(X_train, y_train)
    #print('Metrica del modelo',score)
    scoring={'accuracy':'accuracy','recall':'recall','precision':'precision'}
    scores=cross_validate(gnb,X_train,y_train,cv=kf, scoring=scoring)

    print('Metricas cross_validation',scores)

    #y_pred = gnb.predict(X_test)
    plot_confusion_matrix(gnb,X_test,y_test)
    print(plt.show())
    #print('Accuracy score_gaussian: ', format(accuracy_score(y_test, y_pred)))
    #print('precision score_gaussian: ', format(precision_score(y_test, y_pred)))
    #print('recall score_gaussian: ', format(recall_score(y_test, y_pred)))
    #print('f1 score_gaussian: ', format(f1_score(y_test, y_pred)))


def MLP(X,y):
    X_train, X_test, y_train, y_test = train_test_split (X,y,test_size=0.3,random_state=45)

    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train=scaler.transform(X_train)
    X_test=scaler.transform(X_test)
    kf=KFold(n_splits=5)

    mlp=MLPClassifier(hidden_layer_sizes=(100,100,100), max_iter=9000, alpha=0.0001,
                      solver='adam', random_state=21,tol=0.000000001)
    mlp.fit(X_train,y_train)
    #score= mlp.score(X_train, y_train)
    #print('Metrica del modelo',score)
    scoring={'accuracy':'accuracy','recall':'recall','precision':'precision'}
    scores=cross_validate(mlp,X_train,y_train,cv=kf, scoring=scoring)
    print('Metricas cross_validation',scores)

    #predict=mlp.predict(X_test)
    plot_confusion_matrix(mlp,X_test,y_test)
    print(plt.show())
    #print('Accuracy score_mlp: ', format(accuracy_score(y_test, predict)))
    #print('precision score_mlp: ', format(precision_score(y_test, predict)))
    #print('recall score_mlp: ', format(recall_score(y_test, predict)))
    #print('f1 score_mlp: ', format(f1_score(y_test, predict)))
```

**Figura A1-3. Implementación GaussianNB y MLP.**

```python
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.viridis):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

```

**Figura A1-4. Implementación código matriz confusión para CNN.**

```
177  epoch_num = 5
178  batch_size = 64
179  nb_classes = 2
180
181  def CNN2(X, y):
182      x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=45)
183
184      inputs=np.concatenate((x_train,x_test),axis=0)
185      np.random.seed(1337)
186      Y_train = np_utils.to_categorical(y_train, nb_classes)
187      Y_test=np_utils.to_categorical(y_test, nb_classes)
188      targets=np.concatenate((Y_train,Y_test),axis=0)
189      print('COMENZAMOS ENTRENAMIENTO')
190      acc_per_fold = []
191      loss_per_fold = []
192      num_folds = 2
193      kfold = KFold(n_splits=num_folds, shuffle=True)
194
195      # K-fold Cross Validation model evaluation
196      fold_no = 1
197      for train, test in kfold.split(inputs, targets):
198
199          model = Sequential()
200          model.add(Dense(500, input_shape=(x_train.shape[1],)))
201          model.add(Activation('relu'))
202          model.add(Dropout(0.3))
203          model.add(Dense(250))
204          model.add(Activation('relu'))
205          model.add(Dropout(0.5))
206          model.add(Dense(50))
207          model.add(Activation('relu'))
208          model.add(Dropout(0.5))
209          model.add(Dense(nb_classes))
210          model.add(Activation('sigmoid'))
211
212       # gradient descent
213          sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
214          model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
215          print(model.summary())
216          print(f'Training for fold {fold_no} ...')
217       # Model Training
218          model.fit(inputs[train], targets[train], batch_size=batch_size, epochs=epoch_num, verbose=1)
219       # Model Prediction
220          scores= model.evaluate(inputs[test],targets[test],batch_size=batch_size,verbose=0)
221          print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]};
222                                  {model.metrics_names[1]} of {scores[1]*100}%')
223          Y_test_prueba=np.argmax(targets[test],axis=1)
224          y_test_predclass = model.predict_classes(inputs[test], batch_size=batch_size)
225          print(classification_report(Y_test_prueba,y_test_predclass))
```

**Figura A1-5. Implementación CNN.**

```
227         cm= metrics.confusion_matrix(Y_test_prueba,y_test_predclass)
228         # Plot non-normalized confusion matrix
229         plt.figure()
230         plot_confusion_matrix(cm,classes=['0','1'])
231         print(plt.show())
232         acc_per_fold.append(scores[1] * 100)
233         loss_per_fold.append(scores[0])
234         fold_no = fold_no + 1
235     print('------------------------------------------------------------------------')
236     print('Score per fold')
237     for i in range(0, len(acc_per_fold)):
238         print('------------------------------------------------------------------------')
239         print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy: {acc_per_fold[i]}%')
240     print('------------------------------------------------------------------------')
241     print('Average scores for all folds:')
242     print(f'> Accuracy: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
243     print(f'> Loss: {np.mean(loss_per_fold)}')
244     print('------------------------------------------------------------------------')
245
```

**Figura A1-6. Métricas medias y matriz confusión CNN.**

## 6.3 Preprocesado

```
17  def preprocess_text(s):
18      punctuation = ["!", "¡", "?", "¿", ".", "(", ")", "'",
19                                  ",",">", "<",":", "/", "º", "%", "£",
20                                  "€", "$", "f-","]","[", "‘", "’", "“", "”",
21                                  "«","»", "=", "*", ";","&","+","_", "|",
22                                  "p-","s-","-", "..."]
23      tildes = [("á", "a"), ("é", "e"), ("i", "i"), ("ó", "o"),("ú", "u")]
24      s = s.lower()
25      s = re.sub('\n', '', s)
26      s = re.sub('rt', '', s)
27     # s = re.sub(r'@[\w_]+', '', s)  #eliminar usuarios(PROBAR SIN ESTO)
28      s = re.sub(r'https?://[^\b]+', '', s) #eliminamos urls
29      #Quitar tildes
30      for j in range(len(tildes)):
31          s = re.sub(re.escape(tildes[j][0]), re.escape(tildes[j][1]), s)
32      #Quitar caracteres
33      for j in range(len(punctuation)):
34          s = re.sub(re.escape(punctuation[j]), '', s)
35      s = "".join(s)
36      #s=tokenizen(s)
37      return s
38
39  def tokenizen(s):
40      lista=word_tokenize(s)
41      for w in range(len(lista)):
42          lista[w]=''.join([i for i in lista[w]])
43          if lista[w] in stopwords.words('english'):
44              lista[w]=''
45      s=" ".join(lista)
46      s=s.replace("  ", " ")
47      return s
```

**Figura A1-7. Preprocesado.**

```
filename='csv/Emoji-hate_safe-en.csv'

#Crear dataframe
df=pan.read_csv(filename)
col=['id', 'tweet', 'sentiment']
df=df[col]
df=df[pan.notnull(df['tweet'])]
df.columns=['id', 'tweet', 'sentiment']
df['category_id']=df['sentiment'].factorize()[0]

df['tweet_pre']=df['tweet'].apply(preprocess_text)

```

**Figura A1-8. Aplicación preprocesado datos.**

## 6.4 Extracciones características y concatenado

```
sentiment_analyzer = VS()
#obtenemos caracteristicas
def count_twitter_objs(text_string):
    space_pattern = '\s+'
    mention_regex = '@[\w\-]+'
    hashtag_regex = '#[\w\-]+'
    parsed_text = re.sub(space_pattern, ' ', text_string)
    parsed_text = re.sub(mention_regex, 'MENTIONHERE', parsed_text)
    parsed_text = re.sub(hashtag_regex, 'HASHTAGHERE', parsed_text)
    return(parsed_text.count('MENTIONHERE'),parsed_text.count('HASHTAGHERE'))
#Creacion de diccionario
df_annotated=pan.read_csv('csv\R_annotated_en.csv')
df_annotated_uni=df_annotated['UNICODE'].values.tolist()
df_annotated_tag=df_annotated['TAG'].values.tolist()

dicEmoji= dict(zip(df_annotated_uni,df_annotated_tag))

def count_emoji_objs(text_string):
    space_pattern = '\s+'
    data = regex.findall(r'\X', text_string)
    text_string2=text_string

    for Word in data:
        if any(char in emoji.UNICODE_EMOJI_ENGLISH for char in Word):
            if len(Word)<=1:
                s='U+{:X}'.format(ord(Word))
                if s in dicEmoji:
                    text_string2=re.sub(Word,' @'+dicEmoji[s]+'@ ',text_string2)

    return(text_string2.count(' @NONE@ '),text_string2.count(' @P@ '),text_string2.count(' @N@ '))

def get_features(s):

    sentiment = sentiment_analyzer.polarity_scores(s)

    data_full = s                    #datos completos
    data = preprocess_text(s)        #datos preprocesados
    num_chars=sum(len(w) for w in data)
    num_chars_total=len(s)
    num_terms=len(data_full)
    num_words=len(data)
    twitter_objs = count_twitter_objs(data_full) #Count #, @, and http://
    emojis=count_emoji_objs(data_full)
    analysisPol = TextBlob(s).polarity
    analysisSub = TextBlob(s).subjectivity
    featurs=[num_terms,num_chars_total,num_words,num_chars,sentiment['compound']+1,
                twitter_objs[0],twitter_objs[1],emojis[0],emojis[1],emojis[2],analysisPol+1,analysisSub]
    return featurs
```

**Figura A1-9. Extracción características numéricas y concatenación.**

```
24  #TfidfVectorizer
25  tfi_df=TfidfVectorizer(sublinear_tf=True,analyzer='word', norm='l2', encoding='latin-1',
26                         ngram_range=(1, 2), stop_words='english', min_df=50,max_df=700000)
27  #print('1')
28  tfi_df2=TfidfVectorizer(sublinear_tf=True,analyzer='char', norm='l2', encoding='latin-1',
29                         ngram_range=(2, 2), stop_words='english', min_df=50,max_df=700000)
30  #COUNTVECTOR
31  vectorizer = CountVectorizer()
32  CV = CountVectorizer(ngram_range=(1, 2),min_df=50, max_df=700000)
33  CV2 = CountVectorizer(analyzer='char',ngram_range=(2, 3),min_df=50, max_df=700000)
34
35  def transform_inputs(tweets,tfi_df,tfi_df2,CV,CV2):
36      features=tfi_df.fit_transform(tweets).toarray()
37      print('3')
38      features2=tfi_df2.fit_transform(tweets).toarray()
39      print('4')
40      oth_array=get_oth_features(tweets)
41      print('5')
42      bigram_vectorizer=CV.fit_transform(tweets).toarray()
43      bigram_vectorizer2=CV2.fit_transform(tweets).toarray()
44      M=np.concatenate([features,features2,oth_array,bigram_vectorizer,bigram_vectorizer2],axis=1)
45      return pan.DataFrame(M)
46
```

**Figura A1-10. Extracción características léxicas y concatenación.**

## 6.5 Main

```
49  #print('TRANSFORMANDO....')
50  X=transform_inputs(df.tweet_pre,tfi_df,tfi_df2,CV,CV2)
51  labels=df.category_id
52
53  def main():
54
55      #GRAFICA DE SENTIMIENTO
56       MatplotLib(df)
57       #SVM
58       SVM_algorimt(X, labels)
59
60       #PRUEBA MULTINOMIAL
61       Multinomialnb(X, labels)
62
63       #PRUEBA GAUSSIAN
64       Gaussiannb(X,labels)
65
66       #PRUEBA MLP
67       MLP(X,labels)
68
69       #PRUEBA CNN
70       CNN2(X,labels)
71
72  if __name__ == "__main__":
73       main()
```

**Figura A1-11. Transformación datos y entrenamiento.**