UNIVERSIDAD DE EXTREMADURA

Universidad de Extremadura

Facultad de Ciencias

GRADO EN MATEMÁTICAS

TRABAJO FIN DE GRADO

CLASIFICACIÓN DE EMOCIONES MEDIANTE APRENDIZAJE AUTOMÁTICO

Laura Rodríguez Ropero

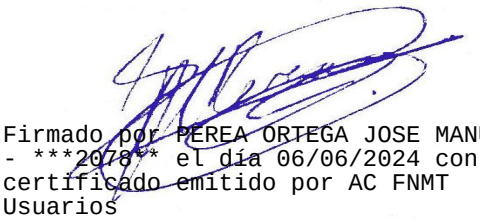
Junio 2024

D. José Manuel Perea Ortega, profesor del Departamento de Ingeniería de Sistemas Informáticos y Telemáticos de la Universidad de Extremadura.

INFORMA:

Que Dña. Laura Rodríguez Ropero ha realizado bajo su dirección el Trabajo Fin de Grado y que la memoria reúne los requisitos necesarios para su evaluación.

Badajoz, 6 de junio de 2024



Firmado por PEREA ORTEGA JOSE MANUEL - ***2078** el día 06/06/2024 con un certificado emitido por AC FNMT Usuarios

Fdo. José Manuel Perea Ortega

Índice general

Resumen

Este Trabajo de Fin de Grado se centra en la implementación de técnicas de inteligencia artificial, con especial énfasis en el aprendizaje automático y las redes neuronales artificiales, para afrontar el reto de categorizar emociones en el ámbito del procesamiento de lenguaje natural. Esta investigación explora la eficacia y precisión de estas tecnologías emergentes al abordar la tarea particular de la clasificación de emociones en textos.

El análisis se adentra en la importancia de identificar y clasificar las emociones humanas presentes en textos breves de redes sociales, tales como ira, asco, miedo, alegría, tristeza o sorpresa, desde una perspectiva matemática. Este enfoque busca no solo comprender las complejidades de la expresión emocional en el lenguaje escrito, sino también evaluar la capacidad de los modelos basados en inteligencia artificial para discernir de manera efectiva estas variadas emociones en contextos cotidianos.

El trabajo comienza con una introducción que contextualiza la relevancia del tema y los objetivos de la investigación. Luego, en el marco teórico, se exploran los conceptos fundamentales y las técnicas empleadas en el procesamiento de lenguaje natural y el aprendizaje automático. A continuación, se detalla la metodología utilizada para la clasificación de emociones mediante modelos de aprendizaje profundo, explicando los algoritmos y las arquitecturas de redes neuronales implementadas. Posteriormente, se presenta una aplicación práctica usando Python, donde se describe el desarrollo y los resultados obtenidos de los experimentos realizados. Finalmente, se concluye con una reflexión sobre los hallazgos, las limitaciones del estudio y las posibles direcciones para futuras investigaciones.

Palabras clave: Inteligencia Artificial, Aprendizaje Automático, Redes neuronales artificiales, Procesamiento de Lenguaje Natural, Emociones

Abstract

This Bachelor's Thesis focuses on the implementation of artificial intelligence techniques, with special emphasis on machine learning and artificial neural networks, to tackle the challenge of categorizing emotions in the field of natural language processing. This research explores the effectiveness and accuracy of these emerging technologies in addressing the specific task of emotion classification in texts.

The analysis delves into the importance of identifying and classifying human emotions present in short texts from social media platforms, such as anger, disgust, fear, joy, sadness, or surprise, from a mathematical perspective. This approach aims not only to understand the complexities of emotional expression in written language but also to evaluate the ability of artificial intelligence-based models to effectively discern these varied emotions in everyday contexts.

The work begins with an introduction that contextualizes the relevance of the topic and the research objectives. Then, in the theoretical framework, fundamental concepts and techniques used in natural language processing and machine learning are explored. Subsequently, the methodology used for emotion classification through deep learning models is detailed, explaining the algorithms and neural network architectures implemented. Following this, a practical application using Python is presented, describing the development and results obtained from the conducted experiments. Finally, the thesis concludes with a reflection on the findings, the limitations of the study, and potential directions for future research.

Keywords: Artificial Intelligence, Machine Learning, Artificial Neural Networks, Natural Language Processing, Emotions

Capítulo 1

Introducción

1.1. Contexto y motivación del trabajo

En el contexto actual, donde la interacción en línea se ha convertido en una parte integral de nuestras vidas, comprender las emociones humanas expresadas en plataformas de redes sociales es más crucial que nunca. La creciente cantidad de información textual generada por los usuarios en estas plataformas presenta un desafío significativo para interpretar y categorizar las emociones de manera eficiente.

Este trabajo se enmarca dentro del campo del Procesamiento del Lenguaje Natural (PLN), una rama esencial de la Inteligencia Artificial (IA). En términos generales, la IA se refiere a la capacidad de las máquinas para llevar a cabo tareas que requieren inteligencia humana. Dentro de este amplio campo, el PLN se ocupa de comprender, interpretar y generar texto de manera análoga a los seres humanos. Esto implica la capacidad de las máquinas para entender la estructura y el significado del lenguaje en diferentes contextos. Ejemplos comunes de aplicaciones del PLN incluyen motores de búsqueda que comprenden consultas en lenguaje natural, asistentes virtuales como Siri o Alexa que responden a comandos verbales, sistemas de traducción automática como Google Translate, análisis de sentimientos en redes sociales para comprender la percepción del público hacia productos o servicios, y generación de texto como la creación automática de resúmenes o la redacción de contenido para sitios web.

El análisis de sentimientos (Sentiment Analysis, SA) o minería de opinión, es una área de investigación dentro del PLN que se enfoca en determinar el tono emocional detrás de un texto. Esta técnica se utiliza ampliamente en la industria para comprender la percepción del público hacia productos, servicios o eventos. Dentro del análisis de sentimientos, una tarea desafiante para la comunidad del PLN es la clasificación de emociones, que implica identificar y categorizar las emociones expresadas en un texto. Esta tarea puede variar desde distinguir entre emociones básicas como felicidad, tristeza o enojo, hasta clasificar emociones más complejas y sutiles. Existen foros de evaluación internacionales como Iberian Languages Evaluation Forum (IberLEF) [1] o Workshop on Computational Approaches to Subjectivity and Sentiment Analysis (WASSA) [2] que proporcionan una visión actualizada de los desafíos y enfoques en esta área.

La clasificación de emociones se puede considerar como una tarea de clasificación de textos, donde el objetivo es asignar una etiqueta emocional a un fragmento de texto. Hoy en día, esta tarea se aborda principalmente mediante métodos estadísticos de aprendizaje automático, que entrenan modelos para reconocer patrones en el texto y asignar la emoción correspondiente.

La motivación de este trabajo radica en la necesidad de aprovechar las herramientas avanzadas de inteligencia artificial y aprendizaje automático para abordar este desafío. La capacidad de las máquinas para comprender las complejidades de la expresión emocional en textos cortos no solo tiene implicaciones en la mejora de la experiencia del usuario, sino que también puede contribuir a la detección temprana de tendencias, crisis o situaciones de emergencia en línea. Otros ejemplos de aplicaciones podrían ser también: conocer la opinión de los clientes de una empresa, la experiencia que han tenido con los productos adquiridos, conocer la opinión de los empleados o para inteligencia de negocio (se utiliza para estimar el impacto que las decisiones y/o acciones de una compañía tendrán en su futuro, el SA permite utilizar datos del pasado para predecir el futuro).

Además, la adopción de una perspectiva matemática en este estudio no solo proporciona un enfoque riguroso, sino que también garantiza una clasificación emocional objetiva y precisa. En última instancia, este trabajo busca contribuir al avance de la inteligencia artificial en la comprensión de las emociones humanas, promoviendo aplicaciones prácticas y éticas en un entorno digital en constante evolución.

1.2. Objetivos

Este trabajo tiene como objetivo abordar el problema de la clasificación de emociones mediante el estudio y la aplicación de técnicas de aprendizaje automático para reconocer y clasificar emociones humanas, tales como alegría, miedo, tristeza o sorpresa, en textos cortos publicados en redes sociales. Para ello, se plantean los siguientes objetivos específicos:

- Revisar los conceptos básicos del aprendizaje automático y redes neuronales artificiales, así como las principales técnicas y herramientas utilizadas para clasificar emociones mediante estas tecnologías.
- Diseñar y entrenar modelos de redes neuronales para clasificar emociones en textos cortos, utilizando diferentes arquitecturas, hiperparámetros y funciones de pérdida.
- Evaluar el rendimiento y la precisión de los modelos propuestos, comparándolos con otros métodos del estado del arte, y analizando sus ventajas y limitaciones.
- Explorar posibles mejoras de los modelos desarrollados.

1.3. Metodología y plan de trabajo

En primer lugar, se realizó una revisión bibliográfica sobre los conceptos básicos del aprendizaje profundo, las redes neuronales artificiales, y su utilización en tareas de procesamiento del lenguaje.

A continuación, se realizó una breve revisión bibliográfica sobre el problema de la clasificación de emociones en textos y se analizó qué modelos de aprendizaje profundo son los más utilizados.

Por último, se desarrollaron algunos de esos modelos para aplicarlos a datos reales, se evaluó su rendimiento y precisión utilizando diferentes métricas (Accuracy y las versiones macro-promediadas de Precission, Recall y F1-score), y se interpretaron los resultados obtenidos comparándolos con otros trabajos previos.

Capítulo 2

Marco teórico

2.1. Inteligencia artificial y aprendizaje automático

La inteligencia artificial (IA) es un campo interdisciplinario de la informática que busca desarrollar sistemas capaces de realizar tareas que, normalmente, requieren de la inteligencia humana. Estos sistemas emplean algoritmos y modelos matemáticos para aprender patrones a partir de datos, adaptarse a nuevas situaciones, y tomar decisiones o ejecutar acciones específicas, todo ello sin intervención humana directa. La inteligencia artificial abarca un espectro amplio de aplicaciones, desde el Procesamiento del Lenguaje Natural hasta la visión por computadora, y se esfuerza por emular no solo la capacidad cognitiva, sino también aspectos emocionales y sociales de la inteligencia humana.

Dentro de este vasto campo de disciplinas y técnicas que constituyen la inteligencia artificial, el **aprendizaje automático** (Machine Learning o ML) se destaca como una subdisciplina fundamental mediante el cuál nuestra máquina aprende a realizar tareas mediante algoritmos basados en modelos estadísticos [3]. A su vez, el **aprendizaje profundo** (DL) es una rama específica del aprendizaje automático basada en el uso de algoritmos capaces de mejorar de forma autónoma gracias a modelizaciones como las redes de neuronas inspiradas en el funcionamiento del cerebro humano y basadas en una gran cantidad de datos [3]. Así, la inteligencia artificial integra tanto al ML como al DL, estructurando un sistema jerárquico donde cada nivel de especialización contribuye al avance de las capacidades cognitivas de las máquinas.

Existen tres tipos de aprendizaje automático [3]:

- El **aprendizaje supervisado**, mediante el cual la máquina adquiere conocimiento a partir de datos previamente etiquetados por humanos. En el caso del reconocimiento de imágenes, por ejemplo, cada imagen utilizada en el proceso de aprendizaje se acompaña de una indicación que especifica su clasificación, conocida como etiquetado.

- El **aprendizaje no supervisado**, en el cual la máquina se dedica a la fragmentación de datos, también conocida como clustering. Implica la acción de distribuir el conjunto de observaciones en grupos pequeños que comparten características comunes. Así, cuando se presenta al algoritmo una nueva observación para clasificar, este se encargará de ubicarla en uno de los grupos que ha identificado. La asignación de etiquetas a los grupos es nuestra responsabilidad, ya que la máquina no tiene conocimiento previo sobre los datos que necesita aprender.

- El **aprendizaje por refuerzo**, mediante el cual la máquina aprende recibiendo recompensas positivas o negativas según las decisiones tomadas.

Las diferencias entre el aprendizaje automático (ML) y el aprendizaje profundo (DL) son principalmente en términos de la arquitectura de los modelos, la representación de características y la capacidad de generalización; mientras que el aprendizaje automático tradicional se basa en modelos más simples y requiere diseño manual de características, el aprendizaje profundo utiliza redes neuronales profundas para aprender automáticamente representaciones de datos complejos, lo que resulta en modelos más flexibles y poderosos, pero también con mayores requisitos computacionales y de datos.

2.2. Redes neuronales artificiales

2.2.1. Fundamentos y tipos

En esencia, una red neuronal artificial es un modelo matemático que trata de actuar como las redes neuronales biológicas, pues está compuesta de pequeñas unidades de procesamiento llamadas neuronas, que se comunican entre sí enviando señales a través de conexiones con pesos asignados [4]. Las neuronas de este tipo de arquitectura están organizadas en capas, que le dan una noción de profundidad (deep).

2.2.1.1. El perceptrón

Fue desarrollado por Frank Rosenblatt en 1957 [5] y representa un algoritmo de aprendizaje supervisado para clasificación. En su forma más básica, el perceptrón también se conoce como **"neurona forma"**, término acuñado a raíz de los trabajos de McCulloch y Pitts en 1943 [6], pues es concebido como un modelo de neurona biológica.
Desde una perspectiva biológica, una neurona se compone de los siguientes elementos [3]:

- Un cuerpo celular, también conocido como pericarion, donde se encuentra el núcleo.
- Múltiples ramificaciones llamadas dendritas, encargadas de recibir la información que va hacia la neurona.
- Un axón que sirve como vía de salida de la información.
- Un recubrimiento de mielina que proporciona protección al axón.
- Terminaciones axonales, también denominadas sinapsis, que establecen conexiones con otras neuronas.

Neurona biológica	**Neurona artificial**
Dendritas	Entradas (input)
Sinapsis	Peso
Axón	Salida (output)
Activación	Función de activación

Tabla 2.1: Comparativa entre una neurona biológica y una neurona formal [3].

Como bien sabemos, la principal función de una neurona biológica es recibir, procesar y transmitir información a través de señales químicas y eléctricas conocidas como impulso nervioso. Estas ondas de naturaleza eléctrica son originadas como consecuencia de un cambio de potencial entre la parte interna y externa de la célula de la siguiente manera: Cuando una neurona recibe múltiples mensajes, realiza la suma de los impulsos nerviosos recibidos. Posteriormente, si esta suma excede un umbral predeterminado, la neurona se activa y transmite un mensaje a través de su axón hacia las neuronas conectadas. Esta secuencia de activación constituye nuestra memoria, de tal manera que para una acción específica un conjunto de neuronas se activa, mientras que otras permanecen inactivas estableciendo así un camino entre la acción y la activación neuronal [3].

A continuación, analizaremos en detalle el funcionamiento del perceptrón simple de capa única, o neurona artificial, cuyo funcionamiento es comparable al de la neurona biológica descrita anteriormente [3]:

En la primera etapa se realiza la suma ponderada de las entradas y se emplea una función de activación para obtener un valor de predicción, se conoce como la **fase de propagación** [3]. Una de las funciones de activación más usadas es la función sigmoide:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Después, comparamos el resultado obtenido por la neurona artificial con la predicción esperada, calculando la diferencia entre el valor esperado y el valor predicho. Este cálculo nos proporciona el **error de predicción**. Una vez que hemos identificado este error, procedemos a retroceder a través de la neurona en sentido inverso para considerar el error cometido durante la predicción y ajustar los valores de los distintos pesos. Esta fase se denomina **retropropagación del error** [3], y para ejecutarla vamos a hacer uso del algoritmo del 'Descenso del gradiente', que explicaremos más adelante.

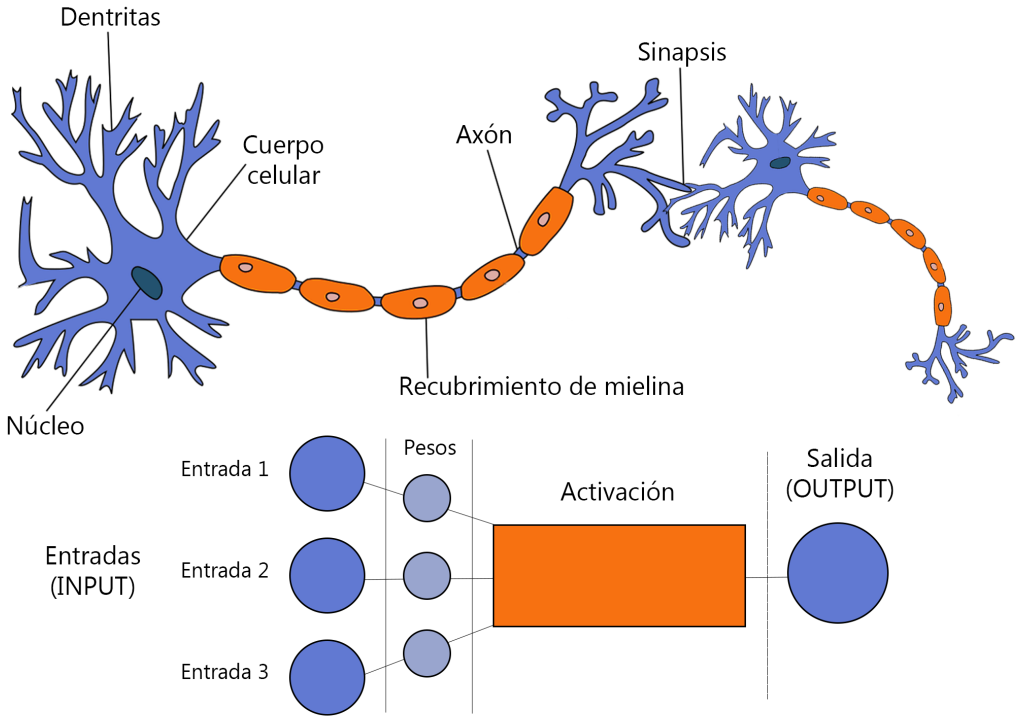


Figura 2.1: Comparativa entre una neurona biológica y una neurona formal.

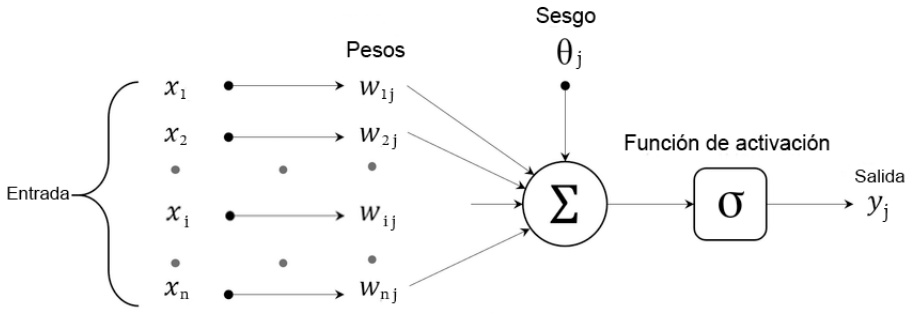


Figura 2.2: Representación de una neurona artificial.

En resumen, una neurona artificial se compone de un conjunto de entradas que representan los datos de una observación (x_i, con $i = 1, .., n$), a cada una de las cuales se le asigna un peso (w_{ij}, con $j = 1, .., n$). Esta pareja entrada/peso permite realizar la fase de propagación usando una función de activación ($\sigma(s) = \frac{1}{1+e^{-s}}$). Ahora, si queremos «forzar» el valor de la predicción para algunos valores de entrada, podemos hacerlo con la ayuda de lo que se denomina un sesgo (θ_j). Entonces la salida del modelo de neurona estándar quedaría de la siguiente forma [4]:

$$y_j = \sigma(\sum_{i=1}^{n} w_{ij}x_i + \theta_j) \tag{2.2}$$

A continuación, se describen algunas de las redes neuronales más utilizadas:

2.2.1.2. El perceptrón multicapa (MLP)

La red neuronal multicapa es una generalización de la red neuronal monocapa, la diferencia reside en que mientras la red neuronal monocapa está compuesta por una capa de neuronas de entrada y una capa de neuronas de salida, esta dispone de un conjunto de capas intermedias (capas ocultas) entre la capa de entrada y la de salida [7]. Gracias a esta arquitectura, es capaz de clasificar datos que no son linealmente separables. Cada neurona en una capa está conectada con todas las neuronas de la capa siguiente (*fully-connected*), formando una estructura de red en la que la información fluye hacia adelante desde la capa de entrada hasta la capa de salida (*feel-forward*). Se emplean algoritmos de aprendizaje supervisado, como retropropagación, para ajustar los pesos de las conexiones y minimizar la diferencia entre las salidas deseadas y las predicciones de la red. Los perceptrones multicapa son efectivos en una amplia gama de aplicaciones de aprendizaje automático, incluyendo clasificación, regresión y reconocimiento de patrones.

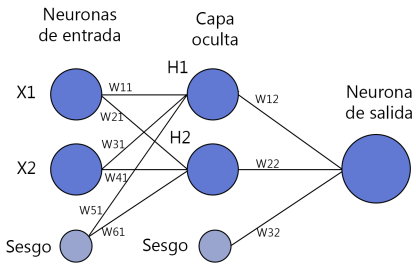


Figura 2.3: Red de neuronas multicapa con 1 capa oculta [3].

2.2.1.3. Red neuronal prealimentada (feed-forward, FFN)

Una red neuronal prealimentada es un tipo básico de red neuronal artificial donde la información fluye en una sola dirección, desde la entrada hasta la salida, sin ciclos ni realimentación. Consiste en capas de neuronas, incluyendo una capa de entrada, una o más capas ocultas y una capa de salida. Cada neurona procesa la información de entrada y la transmite a la siguiente capa, generalmente a través de funciones de activación no lineales. Se utiliza en una variedad de aplicaciones de aprendizaje automático como clasificación de imágenes, reconocimiento de voz y procesamiento del lenguaje natural.

2.2.1.4. Red neuronal recurrente (RNN)

Una red neuronal será recurrente si permite conexiones arbitrarias entre las neuronas, incluso pudiendo crear ciclos (retroalimentación), con esto se consigue crear la temporalidad, permitiendo que la red tenga memoria [7]. Esta capacidad de mantener una memoria a largo plazo es especialmente útil para tareas que involucran datos secuenciales, como el procesamiento del lenguaje natural, la traducción automática y la predicción de series temporales. Una variante de red recurrente es la conocida LSTM, que ya comentaremos más adelante.

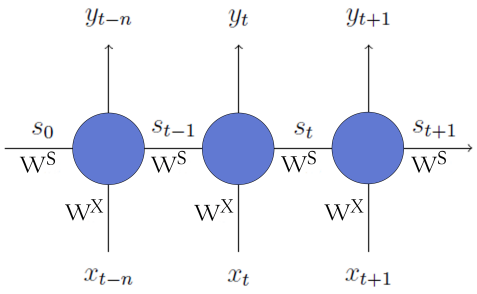


Figura 2.4: Ejemplo de red neuronal recurrente.

2.2.1.5. Red neuronal convolucional (CNN)

Se denomina red convolucional a una red compuesta por capas convolucionales, *pooling* y una capa de reshape, todo ello seguido de un modelo de clasificación, por ejemplo, MLP [8]. Está diseñada especialmente para procesar datos estructurados en forma de grilla, como imágenes. Utiliza las capas de convolución para detectar patrones locales en los datos de entrada, lo que permite aprender características jerárquicas de manera eficiente. La principal diferencia de la red neuronal convolucional con el perceptrón multicapa viene en que cada neurona no se une con todas y cada una de las capas siguientes sino que solo con un subgrupo de ellas (se especializa), con esto se consigue reducir el número de neuronas necesarias y la complejidad computacional necesaria para su ejecución [7]. Las CNN son ampliamente utilizadas en tareas de visión por computadora, como reconocimiento de objetos, detección de rostros y segmentación de imágenes.

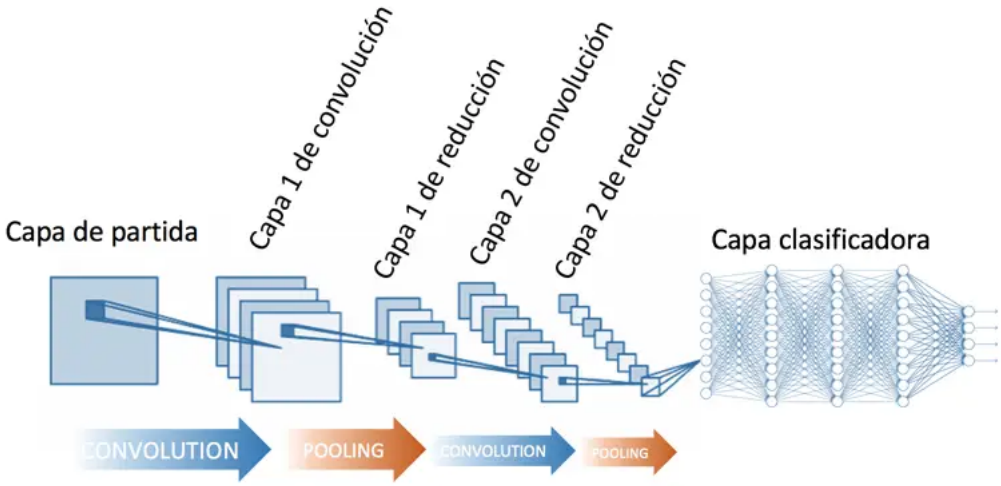


Figura 2.5: Arquitectura de una red neuronal convolucional [7].

Cuando se trata de procesar lenguaje natural o cualquier tipo de datos secuenciales, las redes neuronales convencionales (prealimentadas) no son la mejor opción, ya que no pueden capturar las relaciones temporales entre los elementos de la secuencia. En su lugar, se prefieren las redes neuronales recurrentes (RNN) o las redes neuronales convolucionales (CNN), ya que están diseñadas específicamente para manejar datos secuenciales y aprender representaciones jerárquicas de estos datos.

2.2.2. Redes neuronales recurrentes

Una red neuronal recurrente, también llamada *feed-back*, es aquella que contiene bucles de retroalimentación. El funcionamiento de una red recurrente introduce una nueva variable t, a la cual se suele llamar tiempo aunque no necesariamente se refiera a la magnitud física con ese nombre [4]. Las redes recurrentes cambian sus propiedades respecto a las redes *feed-forward* de la siguiente manera [4]:

1. En lugar de una única matriz de pesos, hay dos (W^X y W^S) que se utilizan en todos los instantes t, lo que significa que no se genera una nueva matriz de pesos para cada instante t.

2. La suma ponderada cambia para tener en cuenta dichas matrices de pesos.

$$s_t = W^X \cdot x_t + W^S \cdot s_{t-1} + \theta_t \tag{2.3}$$

3. La salida al final de la red corresponde a la siguiente expresión:

$$y_t = \sigma(W^X \cdot x_t + W^S \cdot s_{t-1} + \theta_t) \tag{2.4}$$

2.2.2.1. Redes recurrentes simples

Supongamos que la entrada del modelo es una secuencia de n componentes, $X = \{x_1, x_2, ..., x_n\} : x_i \in \mathbb{R}^{d_X}$ y la salida del modelo es una secuencia del mismo tamaño, i.e. $Y = \{y_1, y_2, ..., y_n\} : y_i \in \mathbb{R}^{d_Y}$. De esta forma, para cada componente x_i de una secuencia de entrada (siguiendo un orden de izquierda a derecha), se calcula la correspondiente salida y_i utilizando la función $f(W^Y y_{i-1} + W^X x_{i-1})$, donde W^X es la matriz de pesos que conecta las neuronas de la capa recurrente con la entrada x_i, W^Y es la matriz de pesos que conecta las neuronas de la capa recurrente con la salida y_{i-1} y f es una función de activación [8]. Estos modelos utilizan una especie de memoria para comprender una secuencia temporal. No obstante, esta memoria tiende a debilitarse a medida que la secuencia de entrada se hace más larga, i.e., no es capaz de resolver problemas donde la salida dependa de información muy antigua, ni tampoco de ajustar o determinar el contexto necesario de la secuencia para calcular una determinada salida. Además, aparecen otros problemas como *vanishing gradient* [8]. Para resolverlo se proponen las redes LSTM:

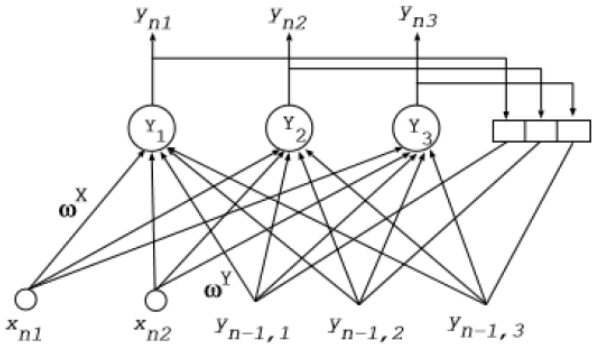


Figura 2.6: Topología de una red recurrente simple para $X = \{x_1, x_2, ..., x_n\} : x_n \in \mathbb{R}^2$ [8].

2.2.2.2. Long Short-Term Memory (LSTM)

Las redes LSTM son capaces de aprender dependencias a largo plazo en secuencias de datos y mantener información relevante durante períodos prolongados de tiempo. Esto es posible gracias a unas unidades de memoria especiales llamadas "celdas de memoria" que poseen una estructura de puertas. Cada celda de memoria tiene tres puertas [8]:

- Puerta de olvido (Forget gate): esta puerta decide qué información almacenada en la celda de memoria debe olvidarse o descartarse.
- Puerta de entrada (Input gate): esta puerta decide qué nueva información se debe agregar a la celda de memoria.
- Puerta de salida (Output gate): esta puerta decide qué información se debe pasar al siguiente paso de tiempo o utilizarse como salida de la celda de memoria.

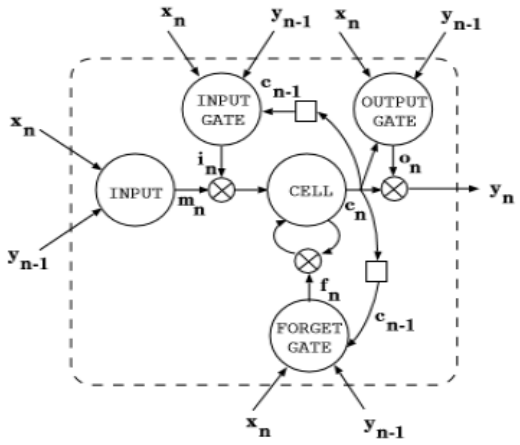


Figura 2.7: Topología de una red LSTM [8].

En la primera celda (input), se determina m_n de manera similar a como se calcula la salida en una red recurrente simple, i.e. $m_n = f(W^X x_n + W^Y y_{n-1})$. Luego, se calcula la salida de la puerta de entrada, $i_n = f(W^Y y_{n-1} + W^X x_n + W^C c_{n-1})$ y se pondera m_n con i_n: $(m_n * i_n)$. Seguidamente, se calcula $c_n = f_n \times c_{n-1} + i_n \times m_n$, haciendo uso de la salida de la puerta de olvido, $f_n = f(W^Y y_{n-1} + W^X x_n + W^C c_{n-1})$. Esto es similar a lo realizado al calcular $m_n * i_n$ y pretende determinar qué cantidad de información de $m_n * i_n$ debe olvidar en función de un cálculo intermedio, c_{n-1}, de la componente anterior de la secuencia. Finalmente, se obtiene la salida de la red, $y_n = o_n \times f(c_n)$, donde o_n es la salida de la puerta de salida, $o_n = f(W^Y y_{n-1} + W^X x_n + W^C c_n)$ y f es una función de activación (usualmente la tangente hiperbólica) [8]. (Ver figura 2.7)

En resumen, las redes LSTM son un tipo de red neuronal recurrente que puede mantener y manipular información a largo plazo en secuencias de datos, lo que las hace especialmente útiles en tareas de modelado de lenguaje, traducción automática, generación de texto y muchas otras aplicaciones que implican el procesamiento de secuencias de datos.

2.2.2.3. Gated Recurrent Unit (GRU)

La mayor diferencia respecto a las LSTM radica en su carga computacional, siendo notablemente más eficientes en términos de tiempo y recursos. A cambio de esta eficiencia, se sacrifica parte de su capacidad representativa. Por consiguiente, cuando lidiamos con secuencias de texto demasiado extensas, las redes GRU no resultan idóneas [9].

2.2.2.4. Redes recurrentes aumentadas (ARNN)

Estas redes están formadas por una estructura que combina una red recurrente simple y una red prealimentada. Se pueden considerar como una extensión de las redes recurrentes estándar. Un ejemplo destacado de este tipo es la red de Elman, compuesta por una red prealimentada de una única capa [8]. (Ver figura 2.8.) Es posible generalizar este tipo de redes mediante la inclusión de nuevas capas en la red prealimentada o de nuevas capas en la red recurrente [8]. En una ARNN típica, la entrada pasa primero por una capa de la RNN, que procesa la secuencia de datos de entrada de manera secuencial y captura las dependencias temporales. Luego, la salida de la RNN se alimenta a una capa de la FFN, que puede realizar operaciones de procesamiento posteriores, como la extracción de características, la reducción de dimensionalidad o la clasificación final.

Esta combinación permite aprovechar las capacidades de modelado secuencial de la RNN junto con las capacidades de aprendizaje no secuencial de la FFN.

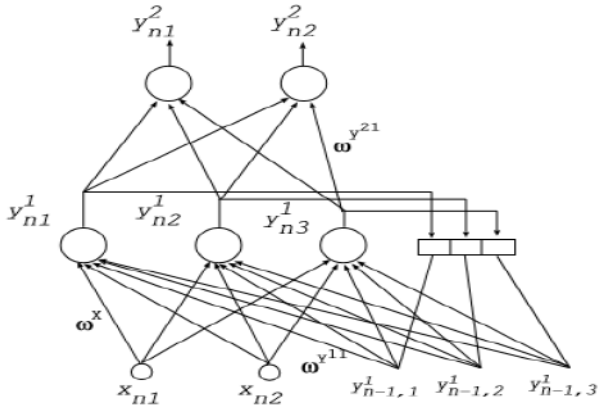


Figura 2.8: Topología de una red de Elman con entrada $x : x_i \in \mathbb{R}^2$ y salida $y : y_i \in \mathbb{R}^2$ [8].

2.2.2.5. Redes recurrentes bidireccionales

Hasta ahora, hemos asumido que las secuencias se procesan en un orden natural, de izquierda a derecha. Esto significa que las redes que hemos mencionado solo pueden considerar relaciones con eventos anteriores en la secuencia. Sin embargo, también puede haber relaciones con los siguientes elementos de una secuencia. Para abordar este tipo de problemas, se han propuesto las redes recurrentes bidireccionales. Estas redes combinan los resultados de procesar las secuencias en ambas direcciones (de izquierda a derecha y de derecha a izquierda). Por ejemplo, en tareas como el análisis de sentimientos, la polaridad de un punto en la secuencia puede depender no solo de los elementos ya procesados, sino también de elementos posteriores [8].

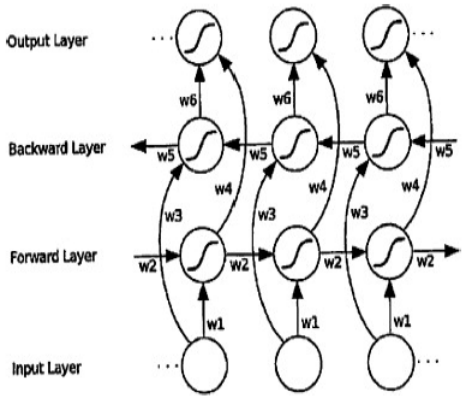


Figura 2.9: Topología de una red recurrente bidireccional [8].

2.2.3. Optimización de una red neuronal

En esta sección, exploraremos los aspectos fundamentales a considerar para optimizar el rendimiento de las redes neuronales. Desde la selección adecuada de funciones de coste y activación hasta el ajuste de hiperparámetros y técnicas avanzadas como el descenso del gradiente y la normalización por lotes, cada paso juega un papel crucial en la mejora de la eficiencia y la precisión de nuestros modelos. Además, abordaremos los desafíos comunes de generalización, como el underfitting y el overfitting, y examinaremos estrategias para mitigar estos problemas, como el dropout y el aumento de datos. A lo largo de esta sección, exploraremos estos conceptos clave y proporcionaremos pautas prácticas para mejorar la capacidad predictiva y la estabilidad de las redes neuronales.

2.2.3.1. Hiperparámetros

En aprendizaje automático, los hiperparámetros son parámetros externos al modelo que tenemos que definir previamente al proceso de entrenamiento. En contraste, los parámetros del modelo son variables internas que se ajustan durante el entrenamiento. Algunos ejemplos de hiperparámetros son: el número de capas de la red, el número de neuronas por capa, la función de activación, la tasa de aprendizaje, o el batch size [9]. A continuación, se van a describir brevemente algunos de ellos:

Funciones de activación. El proceso inicial para decidir si una neurona artificial se activa involucra, la suma de los productos entre los pesos sinápticos y los valores de entrada. Esta fase puede considerarse como una **preactivación de la neurona**. Después de sumar estos productos, se aplica una función que determina si la neurona se activa o no. Esta activación es esencialmente la predicción que hace nuestra neurona [3]. Según Aurélien Vannieuwenhuyze [3], para calcular esta predicción, contamos con varias funciones:

- **Función de umbral binario;** esta función devuelve un valor igual a 0 o 1.
- **Función sigmoide;** esta función devuelve valores entre 0 y 1 más progresivos. El inconveniente es que valores negativos pueden dar lugar a valores positivos.
- **Función tangente hiperbólica (tanH);** esta función se parece a la función sigmoide, pero tiene la ventaja de que para todo valor negativo permanece o se convierte en muy negativa.

- **Función ReLU (Rectified Linear Unit, unidad de rectificación lineal);** si el resultado de la suma ponderada es inferior a 0, la función asigna el valor 0; de lo contrario, toma el valor resultante de la suma. Esta función ofrece ventajas significativas en términos de eficiencia computacional, ya que implica tomar el máximo entre 0 y el valor de la suma ponderada. Una segunda ventaja es su resistencia a la saturación, a diferencia de las funciones sigmoide y tangente hiperbólica, ya que no tiene un límite en la zona positiva. No obstante, su desventaja radica en que cuando se enfrenta a un valor negativo, automáticamente asigna el valor 0, lo que conlleva la falta de aprendizaje en la red. Para abordar este problema, la función Leaky ReLU presenta una solución al multiplicar el valor negativo por 0,01, permitiendo así activar el proceso de aprendizaje en la red. A pesar de este inconveniente, la función ReLU sigue siendo la más utilizada actualmente como función de activación en redes neuronales.

- **Función softMax;** esta función atribuye una probabilidad a cada una de las distintas clases mientras garantiza que la suma de estas probabilidades sea igual a 1.

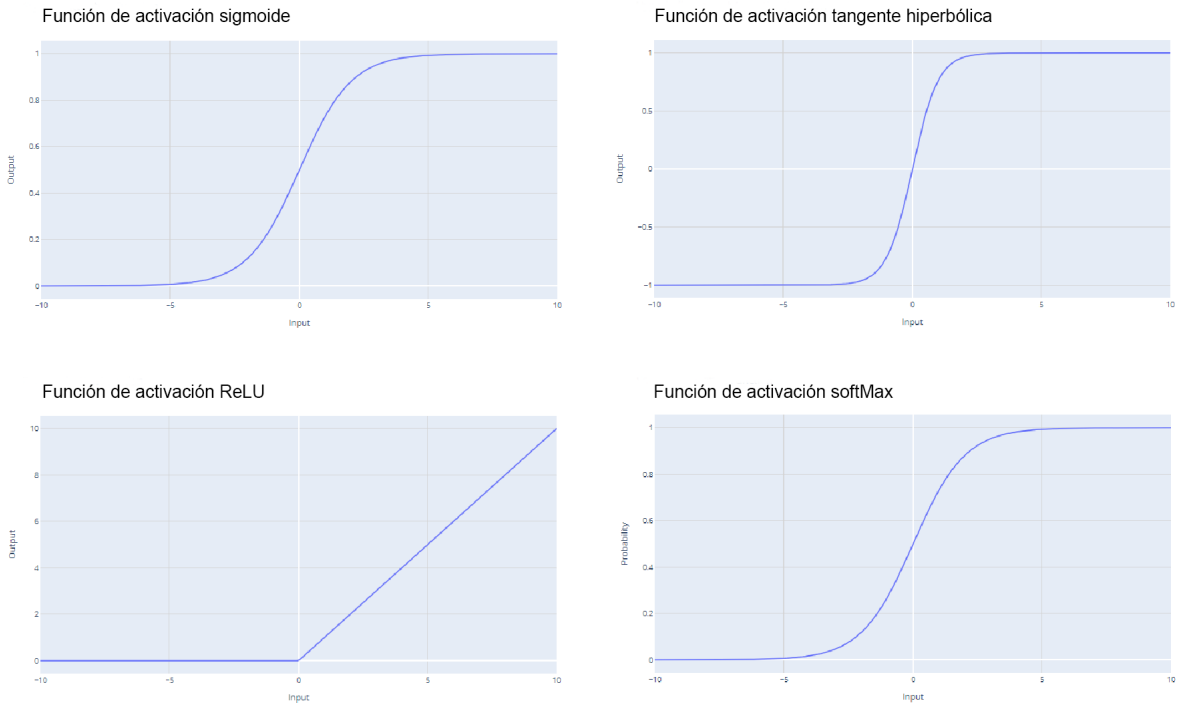


Figura 2.10: Gráficas de las funciones de activación [10].

Como ya se ha comentado antes, existen ciertos tipos de patrones para los que algunos hiperparámetros funcionan mejor o peor con un tipo de problema. Por ejemplo, en el caso de las **funciones de activación**. En la figura 2.11 podemos ver que en el caso de las funciones de activación para capas densas y capas convolucionales dan mejores resultados la función de activación ReLU, mientras que en las capas recurrentes se utilizan las funciones de activación sigmoide y tanH [9].

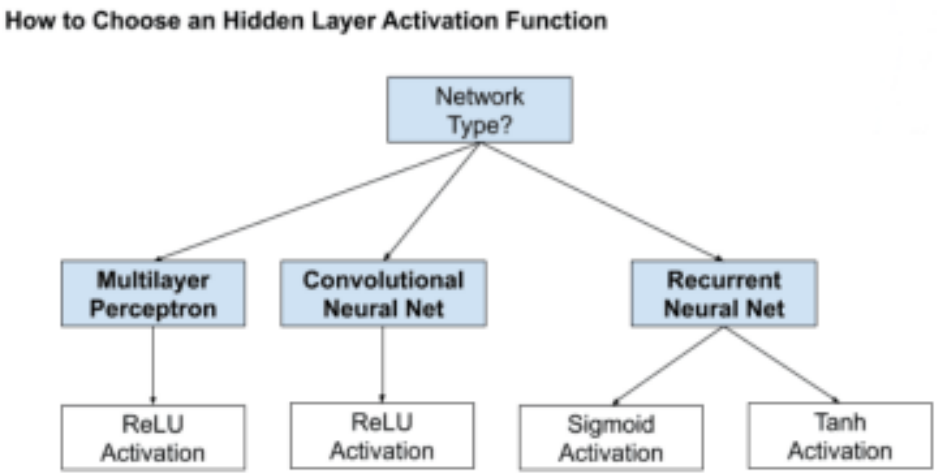


Figura 2.11: Mejores funciones de activación para capas ocultas [9].

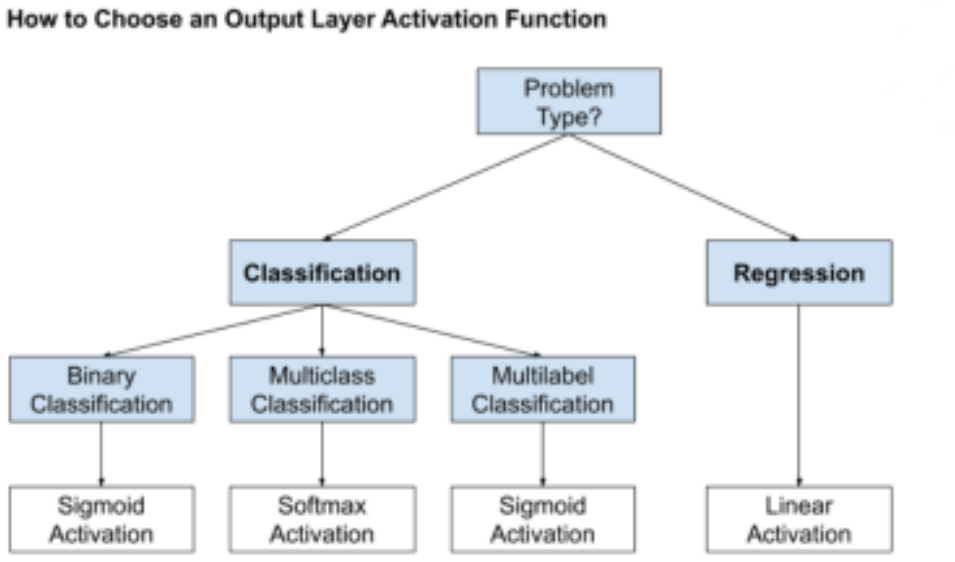


Figura 2.12: Mejores funciones de activación para capas de salida [9].

Funciones de pérdida (Loss function). Se trata de una función que mide la diferencia entre las predicciones de una red neuronal y los valores reales de las observaciones usadas en el entrenamiento. Cuanto menor sea el resultado de esta función, más eficiente será la red neuronal. Minimizar esta función implica ajustar los pesos de la red neuronal [3]. En esencia, nos proporciona una métrica cuantitativa para evaluar el desempeño del modelo y ajustar sus parámetros de manera que se reduzca la pérdida.

Algunas de las funciones de pérdida más utilizadas incluyen [3, 9]:

- **Error cuadrático medio MSE o error global**

$$\mathrm{E} = \mathrm{Prediccion\_real} - \mathrm{Prediccion\_realizada} \tag{2.5}$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(E)^2 \tag{2.6}$$

- **binary_crossentropy**, es la más utilizada en problemas de clasificación donde la entrada se puede clasificar en dos etiquetas.

- **categorical_crossentropy**, es la más utilizada en problemas de clasificación donde la entrada se clasifica en tres etiquetas o más.

Tasa de aprendizaje (learning rate). El descenso del gradiente es un algoritmo que busca minimizar la función de pérdida ajustando gradualmente los pesos de la red. Este ajuste se hace mediante pasos cortos, usando un hiperparámetro denominado tasa de aprendizaje [3]. El objetivo es converger hacia una minimización global de la función de error MSE, que considera el conjunto de observaciones.

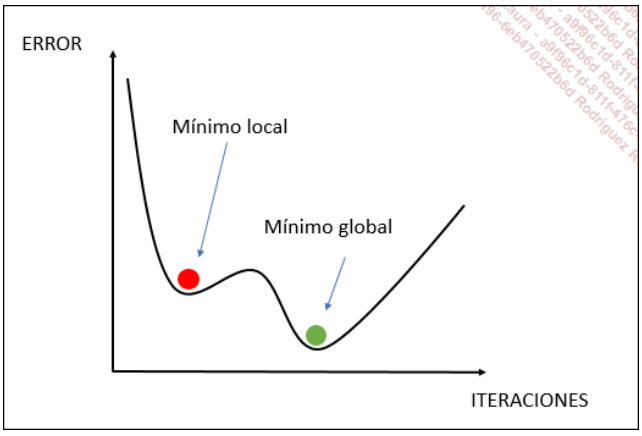


Figura 2.13: Algoritmo descenso del gradiente [3].

Para evitar mínimos locales, la función de error cuadrático medio (MSE) debe ser convexa, lo que garantiza encontrar el mínimo global. Esto es posible siempre y cuando seleccionemos una tasa de aprendizaje adecuada y realicemos suficientes épocas (epoch). Una época se refiere al proceso en el que todo el conjunto de datos se envía a la red neuronal para su procesamiento, experimentando así las fases de propagación y retropropagación [3]. La tasa de aprendizaje es el hiperparámetro fundamental que debemos ajustar en nuestro modelo de red neuronal. Este parámetro regula la velocidad de convergencia de la red, determinando si la convergencia se produce rápidamente o de forma más lenta. Una tasa de aprendizaje elevada facilita que la red neuronal alcance un valor adecuado con un número reducido de iteraciones, aunque puede causar problemas como la convergencia ineficiente. Por otro lado, una tasa de aprendizaje más pequeña permite obtener resultados más precisos, aunque requiere un mayor número de iteraciones para lograr la convergencia [9]. La selección adecuada de este parámetro es esencial para garantizar un entrenamiento efectivo y evitar complicaciones como la falta de convergencia o el sobreajuste.

Batch size. Determina cuántos ejemplos se procesan simultáneamente antes de actualizar los pesos del modelo. Un batch size más grande puede acelerar el entrenamiento, pero puede requerir más memoria y puede ser menos eficiente en términos de generalización del modelo. Por otro lado, un batch size más pequeño puede mejorar la generalización del modelo pero puede ser más lento y requiere más iteraciones para converger.

La elección adecuada de los hiperparámetros puede afectar significativamente el rendimiento y la generalización del modelo, es por eso que se han desarrollado diversas técnicas de búsqueda de los mismos [9]:

- **Grid Search**: esta táctica consiste en generar una malla de n dimensiones, en la cual cada dimensión se asocia con un hiperparámetro. Para cada hiperparámetro (dimensión), se establece un rango de valores y se exploran todas las combinaciones posibles para identificar qué configuración de hiperparámetros proporciona un rendimiento óptimo a la red neuronal.

- **Random Search**: esta táctica tiene una estrategia similar a la anterior. La idea es buscar en menores iteraciones cuáles son los mejores resultados eligiendo los valores de los hiperparámetros aleatoriamente.

2.2.3.2. Desvanecimiento del gradiente o vanishing gradient

Al entrenar redes neuronales mediante *back-propagation* se aplican, de forma iterativa, actualizaciones sobre los pesos en función del gradiente de la función de pérdida con respecto a dichos pesos. Estos gradientes se computan, empezando desde la última capa, mediante la regla de la cadena aplicada sobre cada neurona de cada capa de la red [8]. Si estos gradientes son menores que 1 y/o pasan a través de regiones de baja pendiente en las funciones de activación, pueden volverse exponencialmente pequeños a medida que retroceden hacia las capas iniciales de la red neuronal. Esto puede hacer que las capas iniciales reciban actualizaciones de pesos insignificantes, lo que afecta negativamente el aprendizaje de la red. Veamos algunas posibles soluciones a este problema:

- **Rectified Linear Unit**

  Tradicionalmente, se han utilizado funciones de activación como la sigmoidal o la tangente hiperbólica, cuyas derivadas tienden a ser muy pequeñas en ciertos casos, lo que ocasiona el inconveniente conocido como el desvanecimiento del gradiente. Para abordar esta dificultad, se han propuesto diversas soluciones. Entre ellas, destaca el uso de la función de activación ReLU (Rectified Linear Unit), que resuelve el problema al presentar una derivada constante de 1 cuando su entrada es positiva y 0 en caso contrario. Esto asegura una propagación total del gradiente hacia atrás cuando la entrada es positiva [8]. Sin embargo, a pesar de sus beneficios, ReLU enfrenta un problema conocido como *dying ReLU*, que se produce cuando la mayoría de las preactivaciones son negativas. Para contrarrestar esta situación, se han propuesto funciones alternativas como Leaky ReLU (LReLU) [8].

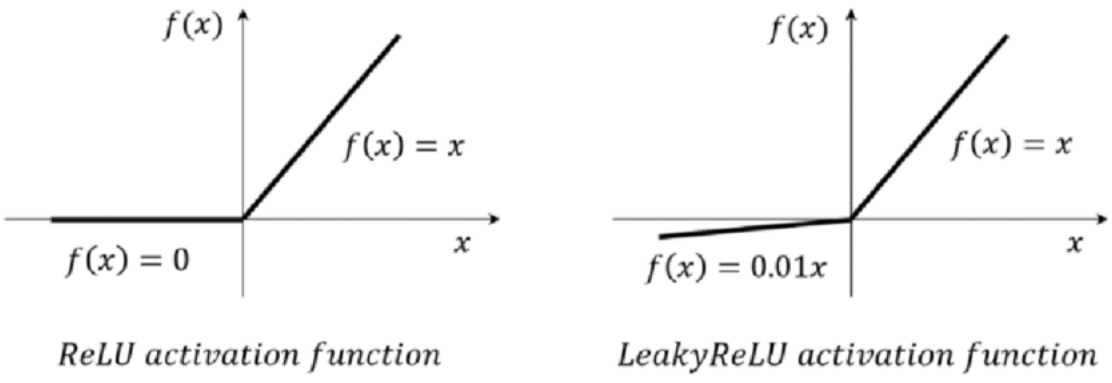


Figura 2.14: ReLU activation function vs. LeakyReLU activation function [11].

- **Batch normalization**

  Con frecuencia, la normalización de datos se realiza solo en la capa de entrada, ajustando las muestras para tener una media de 0 y una desviación estándar de 1. Sin embargo, durante el flujo de datos a través de la red, los pesos modifican los valores, lo que hace que se pierda esta normalización y resulta en una distribución de salida diferente en cada capa, conocida como *Internal Covariance Shift*. Para abordar este problema, se propuso *Batch Normalization*. Esta técnica normaliza los datos en cada mini-batch que fluye a través de la red, asegurando que todas las capas de la red reciban entradas normalizadas. Esto se logra calculando la media y la desviación estándar de cada minibatch y normalizando los datos antes de pasarlos a través de las capas de la red. Además de mantener la estabilidad del gradiente durante el entrenamiento, también puede acelerar el proceso de entrenamiento al permitir el uso de tasas de aprendizaje más altas [8].

2.2.3.3. Problemas de generalización

Debido a la gran cantidad de parámetros que tienen los modelos de deep-learning y, generalmente, a la reducida cantidad de muestras que se disponen para ajustarlos, surgen problemas de generalización [8].

- El subajuste (*underfitting*) ocurre cuando un modelo de aprendizaje automático es demasiado simple para capturar las relaciones subyacentes en los datos. Esto lleva a una alta tasa de error y baja precisión en las predicciones.

- El sobreajuste (*overfitting*) ocurre cuando un modelo de aprendizaje automático se vuelve tan específico a los datos de entrenamiento que no generaliza bien a nuevos datos de prueba.

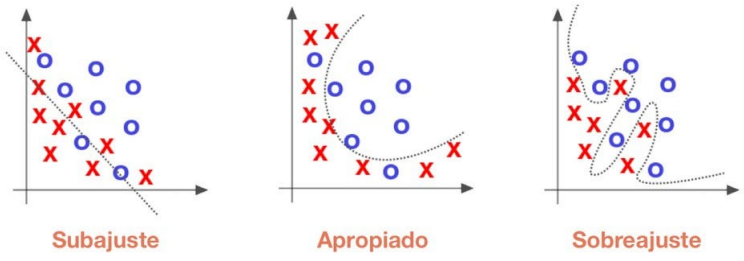


Figura 2.15: Representación gráfica de *underfitting* y *overfitting* [12].

Para abordar el sobreajuste, existen técnicas como:

- **Dropout**, que desactiva aleatoriamente ciertas neuronas para cada mini-batch en cada iteración para reducir la dependencia entre ellas, fomentando una mejor capacidad de trabajo individual [9].
- **EarlyStopping**, que detiene el entrenamiento cuando la pérdida de validación deja de mejorar [9].
- **Data augmentation**, que aumenta la cantidad y la diversidad de los datos de entrenamiento mediante la generación de muestras sintéticas. Este enfoque es especialmente útil cuando hay pocas muestras de entrenamiento en comparación con el número de parámetros del modelo [8].

    - El primer enfoque implica crear muestras sintéticas a partir de datos reales. Al aplicar transformaciones a las muestras originales, se generan nuevas muestras que mantienen las características esenciales de los datos originales, pero introducen variabilidad y diversidad en el conjunto de datos de entrenamiento.
    - El segundo enfoque consiste en la aplicación de ruido (típicamente gaussiano) a los datos existentes.
    - El tercer enfoque, conocido como *Synthetic Minority Over-sampling Technique* (SMOTE), se utiliza específicamente en el contexto de conjuntos de datos desbalanceados. SMOTE genera muestras sintéticas de las clases minoritarias, esto ayuda a abordar el desequilibrio de clases lo que puede mejorar su capacidad para generalizar y predecir con precisión.

2.2.3.4. Problemas adicionales

- El **desbalanceo** entre el número de muestras de las clases de un determinado corpus provoca dificultades a la hora de generalizar en muchos modelos, que dan lugar a un sesgo en las predicciones hacia las clases con mayor probablilidad a priori. Posibles soluciones a este problema pueden ser: incrementar el número de muestras de forma sintética de aquellas clases con menor probabilidad a priori mediante técnicas de *data augmentation*, o escalar la función de pérdida de forma que las clases minoritarias tengan un factor mayor, y así se fuerze a la red a clasificar bien dichas muestras [8].

2.3. Representación de texto en una red neuronal

En el vasto campo del procesamiento del lenguaje natural (PLN), la representación de texto juega un papel fundamental. Se trata de convertir el texto en una forma que las máquinas puedan entender y procesar de manera efectiva. En los últimos años, las redes neuronales han demostrado ser herramientas poderosas para esta tarea, permitiendo capturar la complejidad y riqueza del lenguaje humano.

2.3.1. Procesamiento de secuencias

En esta sección estudiaremos cómo procesar secuencias de elementos (i.e. $x = \{x_1, x_2, ..., x_n\} : x_i \in \mathbb{R}^d$), en vez de vectores de dimensionalidad fija (i.e. $x \in \mathbb{R}^d$), como entrada.

A grandes rasgos se pueden plantear dos estrategias [8]:

- La primera consiste en generar un único vector que encapsule la información de las palabras presentes en el documento, lo que se conoce como **representación vectorial**.
- La segunda estrategia implica representar los documentos mediante una secuencia de elementos que identifiquen las palabras individuales que lo conforman, esta aproximación se denomina **representación secuencial**.

En relación a los tipos de problemas secuenciales que se deben resolver podemos identificar tres categorías distintas según la cardinalidad de la salida esperada [8]:

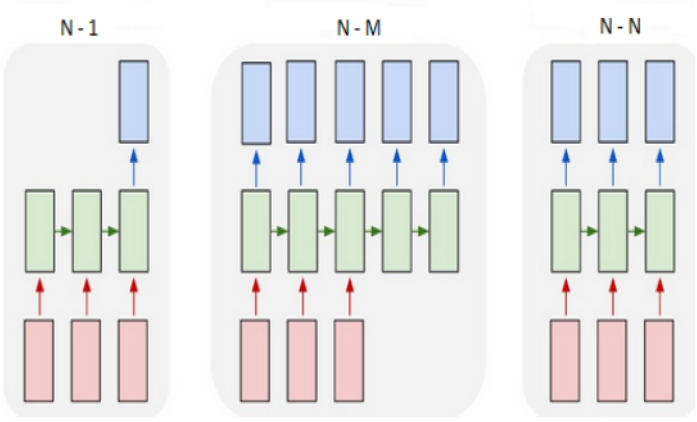


Figura 2.16: Tipos de problemas secuenciales [8].

2.3.2. Representación de constituyentes

Uno de los desafíos clave en las tareas de clasificación de texto es la adecuada representación del mismo. El objetivo principal radica en convertir documentos de texto en una forma numérica que pueda ser procesada computacionalmente. Sin embargo, para lograr esto, es esencial representar de manera efectiva los elementos que conforman dichos documentos, los constituyentes. Algunos de ellos se listan a continuación [8]:

- **Palabras**: son relativamente fáciles de extraer y conllevan un coste espacial bastante reducido en corpus no muy grandes, sin embargo, en corpus de gran tamaño la dimensionalidad de las representaciones puede llegar a ser un problema.
- **Caracteres**: son al igual que las palabras fáciles de extraer y su representación puede ser de dimensionalidad incluso más reducida pues la talla de vocabulario es menor, pero en el caso de representaciones secuenciales el coste espacial es mucho mayor. Además, las relaciones existentes entre caracteres son mucho más complejas que entre palabras.
- **Constituyentes complejos**: la extracción resulta más compleja pero tienen la capacidad de generar información que no está inherentemente presente en los constituyentes originales. Por lo tanto, se consideran representaciones más ricas que proporcionan información adicional a los dos tipos anteriores.
- **n-gramas de constituyentes**: estos constituyentes permiten capturar cierto contexto, al contrario que los constituyentes antes mencionados, pero este contexto suele ser limitado debido al costo espacial que conlleva.

Vamos a mencionar algunas técnicas de representación de constituyentes individuales para, posteriormente, emplearlas mediante otras técnicas de representación a nivel de documento [8].

1. **One-Hot**: cada elemento del vector de representación tiene un valor de 1 para el constituyente correspondiente y 0 para todos los demás constituyentes, lo que garantiza una representación única y distinta para cada uno. Este tipo representación resulta muy sencilla de implementar, pero no es capaz de retener las relaciones entre constituyentes y supone un gran coste espacial.

2. **Word Embeddings**: los constituyentes son mapeados a vectores de números reales. Estos vectores capturan las similitudes semánticas y sintácticas entre las palabras, permitiendo a las máquinas entender el contexto y significado de las palabras en un texto.

    - **Skip-gram (SG)**: se elige una palabra de centro y se intenta predecir las palabras que la rodean dentro de una ventana definida de palabras vecinas. Este enfoque permite capturar las relaciones semánticas y sintácticas entre las palabras, ya que las palabras vecinas tienden a compartir contextos similares. Aunque el modelo Skip-gram es más lento en comparación con CBOW, tiende a capturar mejor la semántica de las palabras, especialmente en conjuntos de datos grandes.
    - **Continuous Bag of Words (CBOW)**: a diferencia del modelo Skip-gram, que predice las palabras vecinas a partir de una palabra de centro, CBOW predice la palabra de centro a partir de las palabras vecinas. Este enfoque permite capturar la información contextual de las palabras circundantes. Es conocido por su eficiencia computacional, especialmente en conjuntos de datos grandes.

3. **Specific Word Embeddings**: los embeddings tienen la capacidad de capturar regularidades lingüísticas basadas en los contextos de los constituyentes, lo que implica que constituyentes similares en contexto tendrán representaciones similares. Sin embargo, este comportamiento puede resultar problemático; por ejemplo, las palabras con polaridades opuestas pueden aparecer en contextos similares, lo que conduce a embeddings similares para palabras con significados completamente diferentes. Por lo tanto, surge la necesidad de especializar los embeddings no solo para capturar contextos similares, sino también para abordar propiedades más específicas como la polaridad. Los embeddings especializados para una tarea determinada se conocen como *Specific Word Embeddings* y requieren datos supervisados para la tarea, lo que representa uno de sus principales inconvenientes [8].

Según José Ángel González Barba [8], existen tres modelos distintos para obtener este tipo de embeddings.

- El modelo **SSWE-Hard** se basa en la polaridad de los documentos, representada por valores discretos, para calcular los embeddings de sus componentes. Aquí, la polaridad del documento se utiliza como única salida para predecir los constituyentes del mismo.

- El modelo **SSWE-Soft**, a diferencia del método previo, emplea probabilidades continuas en lugar de etiquetas discretas para expresar la polaridad de las palabras. Este enfoque es más flexible y puede manejar de manera más efectiva el ruido en las etiquetas, pero aún requiere datos etiquetados.

- El modelo **SSWE-Unified** busca integrar las ventajas de los enfoques Hard y Soft. Combina tanto las etiquetas discretas como las probabilidades continuas para expresar la polaridad de las palabras. Esta estrategia busca aprovechar lo mejor de ambos métodos, ofreciendo una representación robusta de la polaridad de las palabras.

2.3.3. Representación de documentos

El problema consiste en representar cada documento de un conjunto $D = \{d_1, d_2, ..., d_n\}$ como un punto en un espacio métrico S, donde se puede definir una función de distancia entre los elementos de dicho conjunto. Hay dos tipos generales de representaciones [8]:

- Representaciones con pérdida de información temporal y de relaciones entre constituyentes del documento: incluyen métodos como *Bag of Words* (BOW) y colapsado de embeddings. Los modelos de redes neuronales que trabajan con vectores de entrada, como MLP, pueden manejar este tipo de representación.

- Representaciones sin pérdida de información temporal y que mantienen las relaciones entre los constituyentes: se realizan de manera secuencial y requieren el uso de modelos de redes neuronales que manejen secuencias, como RNN o CNN.

Además, estas representaciones no son excluyentes y se pueden combinar. Cada representación puede capturar información diferente, lo que puede proporcionar una comprensión más completa del documento y, posiblemente, simplificar la resolución de las tareas relacionadas [8].

1. **Bag of Words**: cada documento se representa como un vector donde cada elemento del vector corresponde a una palabra del vocabulario, y el valor de cada elemento indica la frecuencia o la presencia de esa palabra en el documento. Esta representación ignora el orden de las palabras en el documento y solo considera su ocurrencia [8].

2. **Colapsado de Embeddings**: lleva a cabo una combinación de embeddings de constituyentes, capaces de capturar gran parte de la semántica presente en dichos documentos. Se diferencian los siguientes tipos [8]:

   - Suma: se suman los vectores de representación de todos los constituyentes que conforman un documento para obtener su representación vectorial continua.
   - Centroide: idéntico a la representación anterior, pero normalizando con respecto al número de constituyentes que se encuentran en el documento y han aparecido durante el entrenamiento del modelo.
   - Palabra próxima al centroide: se considera como representación de la frase el vector de representación del constituyente, presente en el documento, que más próximo se encuentre con respecto al centroide comentado en el punto anterior.
   - Ponderada embeddings con tf-idf: se realiza una combinación lineal de los vectores de constituyentes y un término que permita ponderarlos de una manera determinada. $\text{repr}(s) = \sum_{i=1}^{n} k_i \cdot \text{emb}(w_i)$
   - Producto: se multiplican los vectores de representación de todos los constituyentes que conforman un documento.
   - Derivada: consiste en acumular las derivadas de cada dimensión del vector según la expresión, $\text{repr}(s) = \sum_{j=1}^{|s|-1} (\text{emb}(w_{i+1}) - \text{emb}(w_{i-1}))$.

3. **Representación secuencial**: consiste en concebir un documento como una sucesión de vectores que son la representación de sus constituyentes.
   En este caso, se destaca la capacidad de esta representación para preservar las relaciones temporales entre los elementos de los documentos. Sin embargo, para tareas de clasificación, se requiere considerar relaciones más complejas en comparación con las representaciones globales como Bag of Words (BOW). Por lo tanto, se necesitan conjuntos de datos más grandes para lograr resultados competitivos [8].

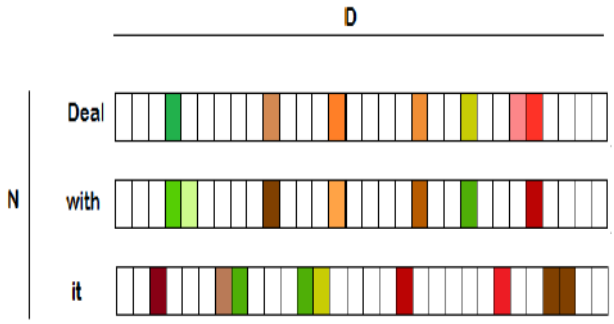


Figura 2.17: Ejemplo de representación secuencial [8].

4. **Combinación de representaciones**: permite aprovechar las fortalezas de cada enfoque y mitigar sus limitaciones individuales. Por ejemplo:

   - Embeddings in-domain, out-domain y lexicones de polaridad.
   - Embeddings ponderados con tf-idf.

Capítulo 3

Clasificación de emociones mediante aprendizaje profundo

La tarea de clasificación de emociones en el procesamiento del lenguaje natural implica asignar una etiqueta emocional a un texto dado. Esto conlleva identificar la emoción predominante expresada en el texto, que puede incluir emociones como felicidad, tristeza, enojo, miedo, sorpresa, entre otras.

Algunos de los enfoques más utilizados en la actualidad para resolver la clasificación de emociones en PLN haciendo uso de aprendizaje profundo incluyen:

- **Redes Neuronales Recurrentes (RNN)**: Las RNNs, en particular las variantes como GRU (Gated Recurrent Unit), son útiles para modelar dependencias temporales en datos de secuencia, lo que las hace efectivas para la clasificación de emociones en texto.
- **Transformers**: Los modelos basados en Transformers, como BERT (Bidirectional Encoder Representations from Transformers); BETO, su variante en español; RoBERTa (Robustly optimized BERT approach); o XLM-RoBERTa (Cross-lingual Language Model from Facebook) han demostrado un rendimiento sobresaliente. Estos modelos capturan el contexto de una manera más amplia y profunda, lo que les permite comprender mejor el tono emocional del texto.

Estos enfoques han sido aplicados en competiciones como IberLEF (2021) [1] o WASSA (2022) [2], donde han destacado por su capacidad para abordar la complejidad y la variabilidad en la expresión emocional en el lenguaje humano.

En los siguientes apartados, vamos a exponer la propuesta actual para la clasificación de emociones mediante aprendizaje profundo poniendo como ejemplo la tarea EmoEvalEs, organizada en IberLEF 2021, en el marco de la 37ª edición de la Conferencia Internacional de la Sociedad Española para el Procesamiento del Lenguaje Natural. El objetivo de esta tarea es promover la Detección y Evaluación de Emociones en Español. Consiste en la clasificación de grano fino de los tweets del corpus EmoEvent en una de las siguientes siete clases: ira, asco, miedo, alegría, tristeza, sorpresa u otro [1].

El enfoque típico para resolver esta tarea en PLN implica varios pasos esenciales que aseguran un análisis y procesamiento efectivo del texto. Primero, se aborda el preprocesamiento y la normalización del texto, un paso crucial que prepara los datos para su análisis posterior, eliminando ruido y estandarizando el formato del texto. Luego, se procede al entrenamiento de una red neuronal, donde se utiliza el conjunto de datos previamente procesado para enseñar al modelo a reconocer patrones y realizar predicciones. Finalmente, se emplean diversas métricas de evaluación para medir la eficacia y precisión del modelo en tareas de clasificación, asegurando que el rendimiento del sistema cumpla con los estándares esperados. En las siguientes secciones, se describirán cada uno de estos pasos en detalle, proporcionando una comprensión profunda de todo el proceso.

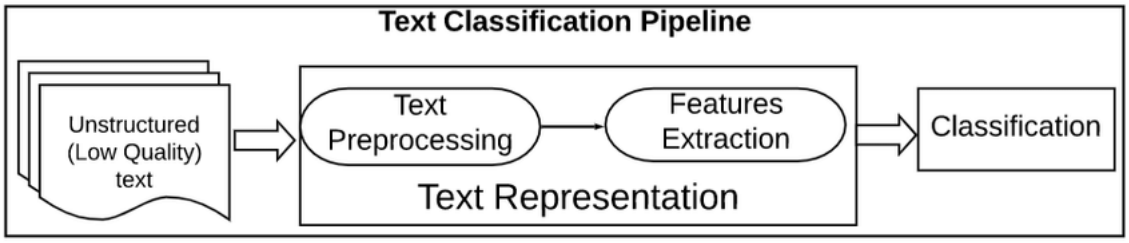


Figura 3.1: Canal de Preprocesamiento de Texto [13].

3.1. Preprocesamiento y normalización del texto

Antes de que los textos puedan ser utilizados eficazmente por algoritmos, es fundamental aplicar una serie de técnicas para preparar los datos. Estas técnicas abarcan desde el preprocesamiento básico, como la tokenización y la eliminación de signos de puntuación, hasta la extracción de características relevantes que puedan ayudar a identificar patrones y contextos emocionales.

El preprocesamiento de datos es el primer paso esencial en el tratamiento de textos en PLN. Los datos de texto no son fáciles de procesar, pues no están estructurados y a menudo contienen mucho ruido. Este ruido puede presentarse en forma de errores ortográficos, errores gramaticales y formato no estándar. Este proceso implica una serie de operaciones diseñadas para limpiar y estructurar los datos textuales de manera que sean adecuados para su análisis por parte de los algoritmos [13]. Según Ricardo Moya [14], existen algunas técnicas comunes para el preprocesamiento de textos, entre las que se incluyen:

- Normalización de texto.
  - Convertir del texto a minúsculas para evitar la distinción entre palabras en mayúsculas y minúsculas.
  - Eliminación de signos de puntuación, como comas, puntos y guiones, pues pueden interferir en el análisis del texto.
  - Eliminación de las storwords, que son palabras comunes que no aportan significado contextual.
- Tokenización del texto, dividir el texto en unidades más pequeñas, generalmente palabras o tokens.
- Eliminación los afijos (sufijos, prefijos, infijos, circunflejos) de una palabra para obtener un tallo de palabra.
- Lematización, proceso lingüístico que sustituye una palabra con forma flexionada (plurales, femeninos, verbos conjugados, etc.) por su lema, es decir, por una palabra válida en el idioma.

Una vez que los datos textuales han sido preprocesados, se seleccionan y extraen características relevantes que puedan ser útiles para identificar la emoción o cualquier otro aspecto de interés en el texto. Según Moez Ali [13], algunos de los métodos más comunes para abordar esta tarea son:

- **Bolsa de palabras (BoW)**: Este método convierte datos de texto en características numéricas. Consiste en construir un vocabulario a partir de las palabras conocidas en el corpus, seguido de la creación de un vector para cada documento que refleje la frecuencia de aparición de cada palabra.

- **TF-IDF ('Term Frequency - Inverse Document Frequency')**: Es otra forma de transformar texto en características numéricas. Cabe mencionar que el modelo TF-IDF considera tanto la frecuencia de las palabras en el documento como su inversa en el conjunto de documentos. Esta consideración hace que el modelo TF-IDF sea más propenso a identificar palabras clave en un documento en comparación con el modelo de Bolsa de Palabras.

Estas técnicas de preprocesamiento y extracción de características son fundamentales para preparar los datos textuales antes de aplicar algoritmos en tareas de clasificación de emociones y otros análisis en PLN.

La biblioteca GSITK es una herramienta versátil diseñada para realizar una amplia variedad de tareas de análisis de sentimientos. En el contexto de este trabajo, se puede utilizar específicamente el módulo 'preprocess' de GSITK para preprocesar un corpus de tweets antes de proceder a la tokenización [15]. Este módulo incluye la interfaz 'Preprocessor', que facilita el uso de la función de procesamiento 'pprocess_twitter'. Esta función está especialmente diseñada para analizar texto de Twitter, extrayendo emojis comunes como tokens especiales y transformando hashtags y menciones para su normalización. De esta manera, GSITK contribuye a preparar los datos de manera efectiva, asegurando que el texto esté en un formato adecuado para el análisis posterior.

3.2. Entrenamiento de una red neuronal

El entrenamiento de una RNN implica alimentar datos secuenciales a la red y ajustar los pesos de las conexiones entre las neuronas para minimizar una función de pérdida específica. Este proceso se realiza típicamente a través de un algoritmo de optimización, como el Descenso de Gradiente. Durante el entrenamiento, la red aprende a capturar patrones y dependencias secuenciales en los datos. Sin embargo, es importante tener en cuenta que en muchos conjuntos de datos puede haber un desbalanceo entre las distintas clases de la variable objetivo, lo cual puede afectar al rendimiento del modelo.

La elección de hiperparámetros adecuados es crucial para el rendimiento óptimo de una RNN. Algunos de los hiperparámetros clave incluyen el tamaño de la capa oculta, la tasa de aprendizaje, el número de capas, la función de activación y la longitud de la secuencia de entrada. La selección de estos hiperparámetros puede influir significativamente en la capacidad de la red para capturar patrones complejos en los datos y evitar problemas como el sobreajuste o el subajuste. Además, existen técnicas de búsqueda de hiperparámetros, desarrolladas en el capítulo anterior, como GridSearch o RandomSearch, que pueden ayudar en la selección de la combinación óptima de los mismos.

Durante el entrenamiento de una RNN, pueden surgir varios desafíos, como el problema de desvanecimiento del gradiente, que dificultan la convergencia del modelo. Además, las RNN pueden sufrir dificultades para capturar dependencias a largo plazo debido a su arquitectura recurrente. Para abordar estos problemas, se han desarrollado varias técnicas, como el uso de capas LSTM (Long Short-Term Memory) o GRU (Gated Recurrent Unit), así como la aplicación de técnicas de regularización como el *dropout* o el *data augmentation*.

Por ejemplo, en la tarea EmoEvalEs del foro de evaluación internacional IberLEF, el equipo GSI-UPM [15] optó por una versión afinada del modelo de lenguaje XLM-Twitter (XLM-T). Este modelo, derivado de XLM-RoBERTa (XLM-R), ha demostrado obtener mejores resultados en el dominio de Twitter gracias a su preentrenamiento con millones de tweets en más de 30 idiomas.

Por otro lado, el equipo BERT4EVER eligió como modelo base a BETO, un modelo BERT entrenado en un corpus en español de gran tamaño. Para abordar problemas como el reducido tamaño del corpus de datos y el desbalanceo entre las proporciones de datos de las distintas emociones, implementaron dos estrategias diferenciadas. Primero, aplicaron un entrenamiento previo continuo, prolongando el entrenamiento sobre un gran corpus de texto sin etiquetar. Segundo, emplearon técnicas de aumento de datos para incrementar la diversidad del corpus de entrenamiento, como la *Back Translation* y el *Mix up*. Estas técnicas no solo aumentaron la diversidad de los datos, sino que también ayudaron a evitar el sobreajuste del modelo [16].

3.3. Métricas de evaluación en clasificación

El proceso de evaluación de un modelo implica el uso de tres conjuntos distintos: el conjunto de entrenamiento, el conjunto de validación y el conjunto de prueba. El modelo se entrena con el conjunto de entrenamiento, se ajustan sus parámetros para minimizar la función de pérdida y se evalúa su rendimiento con el conjunto de validación. La precisión obtenida del conjunto de validación es crucial, ya que la del conjunto de entrenamiento no refleja la capacidad de generalización del modelo. Finalmente, el modelo entrenado se evalúa con el conjunto de prueba para obtener una estimación imparcial de su rendimiento en datos no vistos. Usualmente, se divide el conjunto de datos en un 70-80 % para entrenamiento, un 10-15 % para validación y un 10-15 % para prueba [9].

Existen dos opciones generales para evaluar un modelo [9]:

- La **validación manual**, aunque intuitiva, presenta el problema de la selección adecuada de conjuntos.

- Para abordar esto, se utiliza la **validación cruzada**, siendo el método más común, conocido como k-fold. Este enfoque implica realizar k iteraciones, entrenando y evaluando el modelo k veces sobre el conjunto de entrenamiento. Se generan tres conjuntos: entrenamiento, validación sobre entrenamiento y validación final (prueba), que verifica la generalización del modelo. Este método busca reforzar la independencia de los conjuntos. La precisión final se calcula como la media de las precisiones de las k validaciones realizadas durante el proceso.

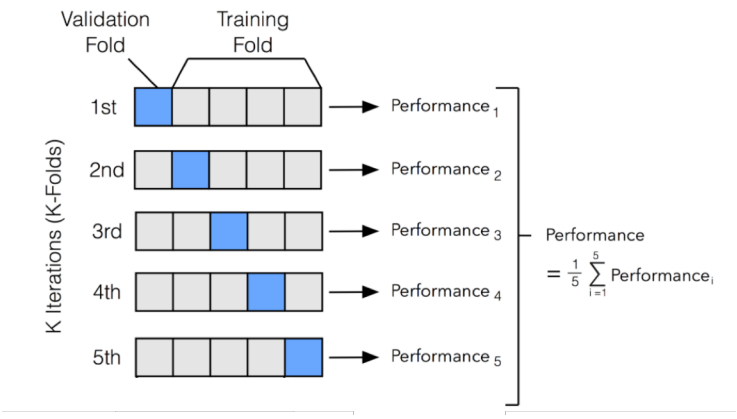


Figura 3.2: Validación cruzada [9].

Las **métricas de rendimiento** existen para evaluar la eficacia y calidad de un modelo o sistema en relación con sus objetivos. Estas métricas proporcionan una manera de cuantificar y comparar el desempeño de un modelo en términos de diversos criterios específicos.

En el contexto del aprendizaje automático y la inteligencia artificial, las métricas de rendimiento son esenciales para evaluar la capacidad predictiva de un modelo. Algunas métricas comunes incluyen [9]:

- **Exactitud (Acuraccy)**: Es la proporción de predicciones correctas (verdaderos positivos más verdaderos negativos) en relación con el total de predicciones. Esta medida es útil en situaciones en las que las clases del modelo están balanceadas.

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

- **Precisión (Precision)**: Es la proporción de instancias positivas identificadas correctamente (verdaderos positivos) en relación con el total de instancias identificadas como positivas (verdaderos positivos más falsos positivos). Es una métrica que se tiene que calcular para cada una de las clases del problema.

$$\text{Precisión} = \frac{TP}{TP + FP} \tag{3.2}$$

- **Exhaustividad (Recall)**: Es la proporción de instancias positivas identificadas correctamente (verdaderos positivos) en relación con el total de instancias positivas en los datos reales (verdaderos positivos más falsos negativos). Es una métrica que se tiene que calcular para cada una de las clases del problema.

$$\text{Exhaustividad} = \frac{TP}{TP + FN} \tag{3.3}$$

- **Valor-F (F-score)**: Es una medida que combina precisión y exhaustividad en una única métrica. Puedes ajustar el parámetro β para controlar el peso relativo de la precisión y la exhaustividad, en la práctica usaremos $\beta = 1$ (*balanced F-score*).

$$F_\beta = \frac{(1 + \beta^2) \cdot \text{Precisión} \cdot \text{Exhaustividad}}{\beta^2 \cdot \text{Precisión} + \text{Exhaustividad}} \tag{3.4}$$

Capítulo 4

Aplicación práctica utilizando Python

En este capítulo, nos adentramos en la aplicación práctica utilizando Python como lenguaje principal. Para llevar a cabo este proceso de manera efectiva, es esencial establecer un marco de trabajo sólido, seleccionar las herramientas adecuadas y aprovechar las librerías disponibles. En este sentido, delineamos el marco de trabajo general para el desarrollo de los modelos, así como las fases clave que conforman este proceso. Además, destacamos las principales herramientas y librerías utilizadas en cada etapa para facilitar la implementación y el análisis de los modelos.

4.1. Descripción del corpus de aprendizaje

El corpus de aprendizaje es la versión en español de un conjunto de tweets basado en eventos que tuvieron lugar en Abril de 2019. Con tweet me refiero a un mensaje corto, con un máximo de 280 caracteres, que puede ser enviado a través de la plataforma de redes sociales Twitter, ahora llamada X. Cada tweet está etiquetado con la emoción principal expresada en el mismo, o en el caso de discrepancia por parte de los anotadores que han etiquetado los tweets, estará etiquetado como 'others' [1].

Los datos se encuentran estructurados en forma de un conjunto de filas, donde cada fila representa un ejemplo de entrada con varias características.

A continuación se muestra un resumen de las características de los ejemplos del corpus de aprendizaje:

- **id:** identificador único asignado a cada ejemplo.
- **event:** el evento o tema al que está relacionado el tweet.
- **tweet:** el contenido del tweet.
- **offensive:** una etiqueta que indica si el tweet contiene lenguaje ofensivo (puede ser 'NO' si no lo contiene o 'OFF' si, en efecto, contiene lenguaje ofensivo).
- **emotion:** una etiqueta que indica la emoción expresada en el tweet de entre las siguientes: 'anger', 'disgust', 'fear', 'joy', 'others', 'sadness', o 'surprise'.

event	**tweet**	**offensive**	**emotion**
NotreDame	#NotreDame La pérdida irreparable que acaba de sufrir la historia de la humanidad ha sido transmitida en vivo a todo el mundo, que desgracia y que pérdida tan grande	NO	sadness
GameOfThrones	El mejor cápitulo de la puta historia. DISFRUTADLO #JuegodeTronos #GameofThrones #BattleOfWinterfell	OFF	joy
SpainElection	Que no nos vais a hacer retroceder, porque cuando te pones las gafas violetas ya no hay marcha atrás #EleccionesA3N #28ALasFeministasVotamos #ElecccionesGenerales28A	NO	others

Tabla 4.1: Algunos ejemplos de tweets del dataset.

Finalmente, los archivos proporcionados que contienen estos datos son tres: el conjunto de entrenamiento (Train), el de validación (Dev), y el de prueba (Test). En la siguiente tabla se puede observar el número de tweets por emoción en cada conjunto, y se puede apreciar a simple vista el desequilibrio entre ellos (desbalanceamiento).

Emotion	Train	Dev	Test
Anger	589	85	168
Disgust	111	16	33
Fear	65	9	21
Joy	1227	181	354
Others	2800	414	814
Sadness	693	104	199
Surprise	238	35	67
Total	5723	844	1656

Tabla 4.2: Distribución de emociones por conjunto.

Realizamos un breve análisis exploratorio de los datos recopilados en el conjunto de entrenamiento utilizando la misma metodología que Joaquín Amat Rodrigo [17]. Inicialmente, llevamos a cabo un recuento del número total de palabras utilizadas en cada categoría de tweets, así como el número de palabras distintas empleadas en cada una de estas categorías (emociones).

Emotion	Nº
Anger	6325
Disgust	1395
Fear	691
Joy	12853
Others	29580
Sadness	7352
Surprise	2588
Total	60784

Tabla 4.3: Palabras totales utilizadas por emoción.

Emotion	Nº
Anger	2546
Disgust	784
Fear	521
Joy	4163
Others	7191
Sadness	2932
Surprise	1418
Total	19555

Tabla 4.4: Palabras distintas utilizadas por emoción.

Este análisis proporciona una visión reveladora de la diversidad léxica en cada categoría, lo que puede sugerir la amplitud temática o la riqueza vocabular de los usuarios en distintos contextos emocionales. Para profundizar en la comprensión de los datos, se han generado gráficos que ilustran las 10 palabras más frecuentes en cada categoría de tweets. A continuación, se muestran ejemplos de estos gráficos:

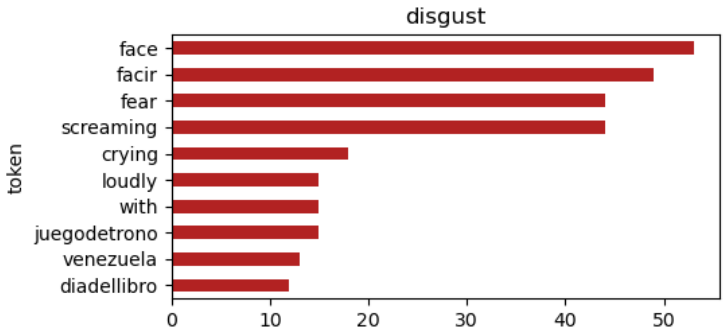


Figura 4.1: Top 10 palabras más usadas por 'disgust'.

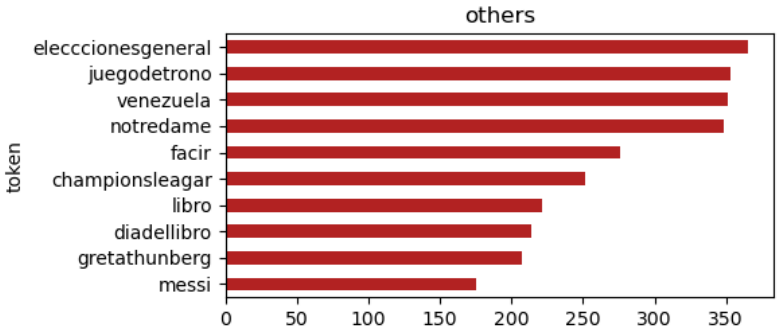


Figura 4.2: Top 10 palabras más usadas por 'others'.

Estos gráficos ofrecen una representación visual clara de las palabras más destacadas en cada emoción, lo que permite identificar las palabras clave y los temas predominantes en cada una. Esta visualización resulta útil para comprender mejor las conversaciones y los discursos presentes en la plataforma, brindando así una comprensión más detallada de los patrones de expresión emocional en el contexto de los tweets analizados.

4.2. Carga de datos

Comenzamos por la lectura de datos desde archivos TSV y la carga de cada conjunto de datos en un DataFrame de 'Pandas'. Para esto haremos uso de librerías fundamentales para la manipulación y el análisis de datos en Python, como son 'NumPy' y 'Pandas'. Posteriormente, eliminamos las filas que contienen valores nulos en cada conjunto de datos utilizando el método 'dropna()'. Luego, se imprime el número de tweets cargados en cada conjunto de datos para verificar el proceso de carga y limpieza de datos. (I.1.2, I.2.2)

4.3. Preprocesamiento y normalización del texto

Antes de utilizar los datos en la construcción de modelos, es necesario realizar una serie de tareas de preprocesamiento. Este proceso se implementa a través de dos funciones:

- **Función limpiar(texto):** Esta función convierte todo el texto a minúsculas, convierte emojis a su significado textual utilizando la librería emoji, y emplea expresiones regulares (regex) de la librería 're' para eliminar enlaces web, menciones de usuarios, signos de puntuación, caracteres especiales, números y espacios en blanco múltiples. Finalmente, retorna el texto limpio. Esta función se aplica a cada tweet en los conjuntos de datos de entrenamiento, validación y prueba mediante el método 'apply()' de Pandas junto con una función lambda.

- **Función normalizar(tweets, min_words=5):** Esta función recibe una lista de tweets junto con sus clases (emociones) y utiliza la librería 'spaCy' para tokenizar los tweets y normalizarlos. 'spaCy' es una librería de procesamiento de lenguaje natural en Python que ofrece funcionalidades avanzadas para el procesamiento de texto. Es particularmente útil para el idioma español debido a sus modelos pre-entrenados de alta calidad y su soporte para diferentes idiomas.

  La función procesa cada tweet convirtiéndolo a minúsculas, eliminando puntos y hashtags, palabras con menos de tres caracteres, signos de puntuación, stopwords, enlaces web y tokens que contengan el carácter " : ". Luego, lematiza las palabras y descarta los tweets que tras la normalización tengan menos de 'min_words' palabras. Finalmente retorna una lista de tweets normalizados junto con sus clases. Esta función se aplica a los conjuntos de datos de entrenamiento, validación y prueba.

En resumen, este código prepara y normaliza los datos de tweets para su posterior procesamiento o análisis, eliminando información no deseada y asegurando que estén en un formato adecuado para su uso. (I.1.3)

4.4. Extracción de características

Después de normalizar los tweets, se procede a guardarlos en archivos binarios utilizando la librería 'pickle'. Estos archivos almacenan los datos preprocesados, lo que evita tener que repetir el proceso de preprocesamiento en cada uso del modelo.

Luego, se cargan los datos normalizados desde los archivos binarios utilizando 'pickle.load()'. Además, se convierten los tweets normalizados en una representación de Bolsa de Palabras (Bag of Words) utilizando 'TfidfVectorizer' de la librería 'scikit-learn' (I.1.4). Este vectorizador transforma los textos en vectores numéricos que representan la frecuencia de las palabras en cada documento, limitándose a 1500 características. Es importante destacar que 'scikit-learn' es una de las librerías más populares para el aprendizaje automático en Python, que ofrece una amplia gama de algoritmos tanto supervisados como no supervisados, así como herramientas para la evaluación y validación de modelos.

4.5. Modelado

En esta sección del código, se definen varios clasificadores, entre ellos un modelo de aprendizaje profundo, XLM-RoBERTa, pre-entrenado e importado desde la biblioteca Transformers de la plataforma Hugging Face. Todo este proceso se realiza en PyTorch, que actúa como el motor subyacente para la computación de estos modelos. Además de este modelo de aprendizaje profundo, se incluyen clasificadores de aprendizaje automático como Multinomial Naive Bayes, Bernoulli Naive Bayes, Support Vector Machines (SVM) con diferentes kernels (lineal, polinómico, rbf y sigmoid), y dos variantes de Random Forest con diferentes profundidades máximas. (I.1.5, I.2.3)

Estos clasificadores se almacenan en un diccionario, donde se asigna a cada clasificador un nombre clave para su identificación. Luego, se itera sobre este diccionario para entrenar cada modelo utilizando los datos de entrenamiento mediante la función 'v.fit(X_train, y_train)', y se calcula la exactitud (accuracy) para estos datos. Finalmente, cada modelo ajustado se guarda en un diccionario que contiene su nombre y el modelo ajustado en sí.

4.6. Evaluación de modelos

Una vez que el modelo ha sido entrenado, se evalúa su desempeño con el conjunto de validación. Para este propósito, se define una función que calcula métricas de evaluación en modelos de clasificación, y las guarda en una lista (I.1.6).

Luego se usan las predicciones de los modelos para alimentar a la función 'classification_report' de la librería 'scikit-learn' y conseguir las tablas que se exponen como resultado en esta sección (I.1.9, I.2.4).

- **Función evaluation(model, name, X_train, y_train, X_dev , y_dev):** Esta función toma como argumentos el modelo, su nombre, así como el conjunto de datos de entrenamiento y el de validación. Calcula las métricas de rendimiento (Accuracy y las versiones macro-promediadas de Precission, Recall y F1-score), utilizando las funciones proporcionadas por 'scikit-learn': 'accuracy_score', 'precision_score', 'recall_score' y 'f1_score'. La salida de la función consiste en un diccionario que contiene el nombre del modelo y los valores de las métricas.

La Accuracy se elegió como la métrica principal para clasificar los sistemas, de la misma forma que hacen en la tarea EmoEvalEs, organizada en IberLEF 2021. Así, analizando los resultados del primer programa, podemos ver que los métodos que demuestran un rendimiento destacado son los SVM con kernels lineal, rbf y sigmoide, y el Random Forest con una profundidad máxima de 50, por ello vamos a descartar el resto de métodos y a trabajar únicamente con estos cuatro.

Tras el entrenamiento de los métodos elegidos, realizamos una primera evaluación con el conjunto de datos de validación, cuyos resultados se encuentran en las tablas 4.5, 4.6, 4.7, y 4.8. A continuación, una vez obtenidas las versiones definitivas de los cuatro modelos, llevamos a cabo una evaluación final utilizando el conjunto de datos de prueba, que consiste en datos completamente nuevos para estos algoritmos. Los resultados de esta última evaluación se detallan en las tablas 4.9, 4.10, 4.11, y 4.12. Estas tablas proporcionan una visión exhaustiva del rendimiento de cada modelo en condiciones reales, lo que valida la robustez y eficacia de los enfoques desarrollados en este estudio.

	P	**R**	**F1**
Anger	0.71	0.38	0.49
Disgust	1.00	0.06	0.12
Fear	0.75	0.33	0.46
Joy	0.63	0.40	0.49
Others	0.63	0.92	0.75
Sadness	0.93	0.60	0.73
Surprise	1.00	0.03	0.06
Accuracy			**0.66**
Macro-avg	**0.81**	**0.39**	**0.44**
Weighted-avg	0.70	0.66	0.62

Tabla 4.5: Classification report dev set SVM lineal kernel.

	P	**R**	**F1**
Anger	0.68	0.35	0.47
Disgust	1.00	0.06	0.12
Fear	0.75	0.33	0.46
Joy	0.60	0.41	0.49
Others	0.62	0.90	0.74
Sadness	0.87	0.60	0.71
Surprise	1.00	0.09	0.16
Accuracy			**0.65**
Macro-avg	**0.79**	**0.39**	**0.45**
Weighted-avg	0.68	0.65	0.61

Tabla 4.6: Classification report dev set SVM rbf kernel.

	P	R	F1
Anger	0.77	0.24	0.36
Disgust	0.00	0.00	0.00
Fear	1.00	0.11	0.20
Joy	0.67	0.38	0.48
Others	0.60	0.93	0.73
Sadness	0.89	0.62	0.73
Surprise	0.00	0.00	0.00
Accuracy			**0.64**
Macro-avg	**0.56**	**0.32**	**0.36**
Weighted-avg	0.64	0.64	0.59

Tabla 4.7: Classification report dev set SVM sigmoid kernel.

	P	R	F1
Anger	1.00	0.16	0.28
Disgust	1.00	0.06	0.12
Fear	1.00	0.11	0.20
Joy	0.70	0.29	0.41
Others	0.59	0.95	0.73
Sadness	0.71	0.62	0.66
Surprise	0.00	0.00	0.00
Accuracy			**0.62**
Macro-avg	**0.72**	**0.31**	**0.34**
Weighted-avg	0.66	0.62	0.56

Tabla 4.8: Classification report dev set Random Forest d=50.

	P	R	F1
Anger	0.12	0.06	0.08
Disgust	0.00	0.00	0.00
Fear	0.00	0.00	0.00
Joy	0.23	0.08	0.12
Others	0.49	0.85	0.62
Sadness	0.16	0.03	0.05
Surprise	0.00	0.00	0.00
Accuracy			**0.45**
Macro-avg	**0.14**	**0.15**	**0.13**
Weighted-avg	0.32	0.45	0.35

Tabla 4.9: Classification report test set SVM lineal kernel.

	P	R	F1
Anger	0.10	0.05	0.07
Disgust	0.00	0.00	0.00
Fear	0.00	0.00	0.00
Joy	0.23	0.09	0.13
Others	0.49	0.83	0.62
Sadness	0.13	0.03	0.05
Surprise	0.00	0.00	0.00
Accuracy			**0.44**
Macro-avg	**0.14**	**0.14**	**0.12**
Weighted-avg	0.32	0.44	0.34

Tabla 4.10: Classification report test set SVM rbf kernel.

	P	R	F1
Anger	0.20	0.02	0.03
Disgust	0.00	0.00	0.00
Fear	0.00	0.00	0.00
Joy	0.06	0.01	0.01
Others	0.49	0.96	0.65
Sadness	0.21	0.02	0.03
Surprise	0.00	0.00	0.00
Accuracy			**0.48**
Macro-avg	**0.14**	**0.14**	**0.10**
Weighted-avg	0.30	0.48	0.33

Tabla 4.11: Classification report test set SVM sigmoid kernel.

	P	R	F1
Anger	0.12	0.01	0.01
Disgust	0.00	0.00	0.00
Fear	0.00	0.00	0.00
Joy	0.13	0.01	0.02
Others	0.49	0.96	0.65
Sadness	0.12	0.02	0.03
Surprise	0.00	0.00	0.00
Accuracy			**0.48**
Macro-avg	**0.12**	**0.14**	**0.10**
Weighted-avg	0.30	0.48	0.33

Tabla 4.12: Classification report test set Random Forest d=50.

El segundo programa utiliza XLM-RoBERTa, un modelo Transformer entrenado en un corpus multilingüe de tweets, lo que elimina la necesidad de llevar a cabo una fase de entrenamiento adicional.

Para comprender mejor el futuro rendimiento del clasificador XLM-RoBERTa en el conjunto de prueba, examinamos detalladamente la matriz de confusión presentada en la figura 5.1. Esta matriz de confusión se genera con ayuda de la función 'confusion_matrix' de la librería 'scikit-learn' en conjunto con la librería 'matplotlib', para graficarla. Esta visualización nos permite identificar patrones y desafíos específicos en la clasificación de las emociones. Se evidencia un marcado desequilibrio en los datos, con casi la mitad de los registros asignados a la clase 'others'. Además, se destaca la notable tendencia de esta clase a confundirse con la clase 'joy'. La matriz también revela la dificultad inherente para distinguir entre emociones que comparten características similares, como 'anger' y 'disgust'. Por último, la escasez de registros en algunas clases, como 'disgust', 'fear' y 'surprise', representa un obstáculo adicional para una clasificación precisa, ya que los modelos tienden a enfrentar dificultades en estas situaciones.

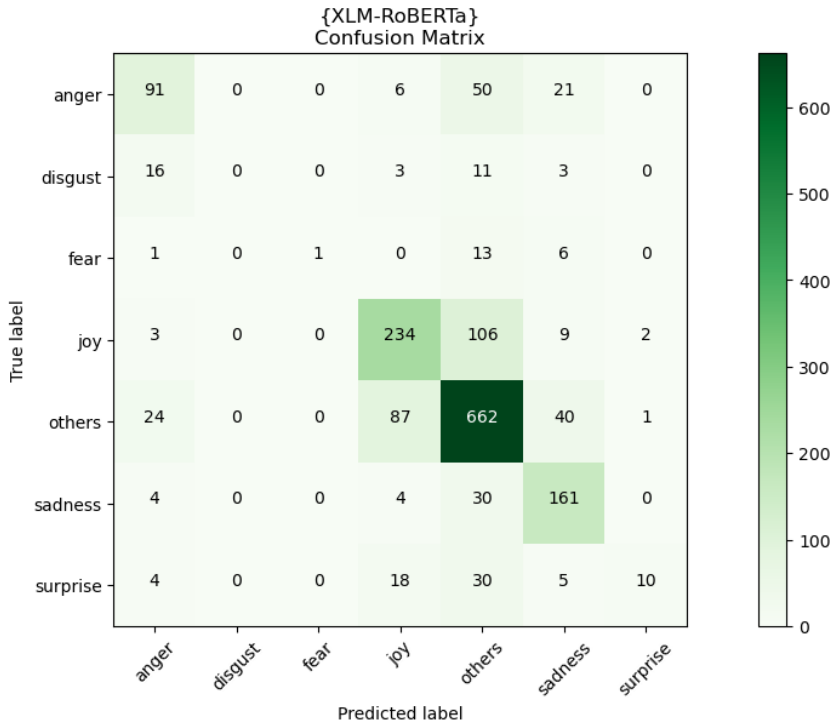


Figura 4.3: Matriz de confusión del modelo XLM-RoBERTa.

Finalmente, se lleva a cabo una última evaluación con el conjunto de datos de prueba. Los resultados de esta evaluación final se presentan en la tabla siguiente 4.13.

	Precision	**Recall**	**F1-score**
Anger	0.64	0.54	0.59
Disgust	0.00	0.00	0.00
Fear	1.00	0.05	0.09
Joy	0.66	0.66	0.66
Others	0.73	0.81	0.77
Sadness	0.66	0.81	0.73
Surprise	0.77	0.15	0.25
Accuracy			**0.70**
Macro-average	**0.64**	**0.43**	**0.44**
Weighted-average	0.69	0.70	0.68

Tabla 4.13: Classification report test set XLM-RoBERTa.

Capítulo 5

Conclusiones

En este estudio, se ha explorado el análisis de sentimientos en el marco del procesamiento de lenguaje natural utilizando una variedad de clasificadores de ML. Se diseñaron dos programas diferenciados. El primero abordó algoritmos tradicionales, como Multinomial Naive Bayes, Bernoulli Naive Bayes, Support Vector Machines (SVM) con diferentes kernels (lineal, polinómico, rbf y sigmoide), junto con dos variantes de Random Forest con diferentes profundidades máximas. Mientras tanto, en el segundo se incorporó un enfoque de aprendizaje profundo mediante XLM-RoBERTa, un modelo pre-entrenado importado desde la biblioteca Transformers de la plataforma Hugging Face.

Los resultados obtenidos revelan que los modelos de ML convencionales no lograron alcanzar un rendimiento satisfactorio en términos de precisión. Los Accuracy finales de estos modelos no superaron el 0.5, lo que indica que tienen dificultades para capturar la complejidad de los datos de texto y las sutilezas en la expresión de sentimientos.

En contraste, el modelo de aprendizaje profundo XLM-RoBERTa demostró un rendimiento significativamente mejor, con un Accuracy final de 0.7. Este resultado sugiere que los modelos basados en Transformers, especialmente aquellos pre-entrenados en grandes corpus de datos, son más efectivos para capturar las características semánticas y contextuales necesarias para el análisis de sentimientos en texto.

Llegados a este punto, es pertinente reflexionar sobre las posibles mejoras que podrían implementarse en futuras investigaciones para optimizar el rendimiento de los modelos propuestos. Sin lugar a dudas, uno de los problemas clave ha sido el desequilibrio de clases, lo que ha provocado un posible sobreajuste durante el entrenamiento y, por ende, un deficiente rendimiento de generalización al ser evaluado en el conjunto de prueba. Para abordar este desafío y mejorar la capacidad de generalización del modelo, se sugiere explorar estrategias de *Data augmentation* que ayuden a mitigar el desequilibrio de clases. Además, podría considerarse la utilización de arquitecturas de modelos más complejas o la optimización de hiperparámetros para ajustar adecuadamente la capacidad del modelo y evitar el sobreajuste. Estas acciones podrían contribuir significativamente a la robustez y eficacia de los modelos en futuros estudios.

Estos hallazgos resaltan la importancia de considerar enfoques de vanguardia en el análisis de sentimientos, especialmente aquellos basados en técnicas de aprendizaje profundo. Además, subrayan la necesidad de continuar investigando y desarrollando modelos que puedan mejorar la comprensión y el procesamiento del lenguaje natural en tareas como la clasificación de sentimientos.

Bibliografía

[1] F. M. P. del-Arco y Salud María Jiménez-Zafra y Arturo Montejo-Ráez y M. Dolores Molina-González y L. Alfonso Ureña-López y M. Teresa Martín-Valdivia, "Overview of the emoevales task on emotion detection for spanish at iberlef 2021," *Procesamiento del Lenguaje Natural*, vol. 67, no. 0, pp. 155–161, 2021. [Online]. Available: http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6385

[2] V. Barriere, S. Tafreshi, J. Sedoc, and S. Alqahtani, "Wassa 2022 shared task: Predicting empathy, emotion and personality in reaction to news stories," in *Proceedings of the 12th Workshop on Computational Approaches to Subjectivity, Sentiment & Social Media Analysis*, J. Barnes, O. De Clercq, V. Barriere, S. Tafreshi, S. Alqahtani, J. Sedoc, R. Klinger, and A. Balahur, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 214–227. [Online]. Available: https://aclanthology.org/2022.wassa-1.20

[3] A. Vannieuwenhuyze, "Inteligencia artificial fácil: machine learning y deep learning prácticos," 2020. [Online]. Available: https://www.eni-training.com/portal/client/mediabook/home

[4] A. P. Sanjuán, "Aplicación para el análisis de sentimientos y tendencias en redes sociales," Trabajo Fin de Grado. Universitat Politécnica de Valencia, 2019.

[5] F. Rosenblatt, "The perceptron - a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, Ithaca, New York, Tech. Rep. 85-460-1, January 1957.

[6] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5(4), pp. 115–133, 1943.

[7] D. Calvo, "Clasificación de redes neuronales artificiales," 2017. [Online]. Available: https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[8] J. Ángel González Barba, "Aprendizaje profundo para el procesamiento del lenguaje natural," Trabajo Fin de Máster. Universitat Politécnica de Valencia, 2017.

[9] J. H. Llanos, "Análisis de sentimientos en twitter mediante técnicas de deep learning," Trabajo Fin de Grado. Universidad de Valladolid, 2022.

[10] M. Ali, "Introducción a las funciones de activación en las redes neuronales," November 2024. [Online]. Available: https://www.datacamp.com/es/tutorial/introduction-to-activation-functions-in-neural-networks

[11] "A generative adversarial network model for design and discovery of multi principal element alloys - scientific figure on researchgate," February 2022. [Online]. Available: https://www.researchgate.net/figure/ReLU-activation-function-vs-LeakyReLU-activation-function_fig2_358306930

[12] L. González, "Overfitting y underfitting," June 2019. [Online]. Available: https://aprendeia.com/sobreajuste-y-subajuste-en-machine-learning/

[13] M. Ali, "Understanding text classification in python," November 2022. [Online]. Available: https://www.datacamp.com/tutorial/text-classification-python

[14] R. M. García, "Procesamiento de lenguaje natural, con python," 2021-2022. [Online]. Available: https://github.com/RicardoMoya/NLP_with_Python/tree/master

[15] D. Vera, O. Araque, and C. Iglesias, "GSI-UPM at IberLEF2021: Emotion Analysis of Spanish Tweets by Fine-tuning the XLM-RoBERTa Language Model," in *Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2021). CEUR Workshop Proceedings, CEUR-WS, Málaga, Spain*, 2021.

[16] Y. Fu, Z. Yang, N. Lin, L. Wang, and F. Chen, "Sentiment analysis for spanish tweets based on continual pre-training and data augmentation," in *IberLEF@SEPLN*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:238207494

[17] J. A. Rodrigo, "Análisis de texto (text mining) con python," December 2020. [Online]. Available: https://cienciadedatos.net/documentos/py25-text-mining-python

Anexos

Anexo I

Código desarrollado en Python

I.1. Algoritmo Machine Learning

I.1.1. Librerías

```
import pandas as pd
import numpy as np
import re, spacy, emoji
from tqdm import tqdm
nlp = spacy.load('es_core_news_sm')
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import itertools
import warnings
warnings.filterwarnings('ignore')
import pickle
```

I.1.2. Carga de datos

```
# Leemos el fichero y lo pasamos a un DataFrame
tweets_train = pd.read_csv('./data/emoevales_train.tsv', sep='\t+')
tweets_dev = pd.read_csv('./data/emoevales_dev.tsv', sep='\t+')
tweets_test = pd.read_csv('./data/emoevales_test.tsv', sep='\t+')
test_gold = pd.read_csv('./data/emoevales_test_gold.tsv', sep='\t+')

# Eliminamos los tweets que tengan algún valor nulo
tweets_train = tweets_train.dropna()
tweets_dev = tweets_dev.dropna()
tweets_test = tweets_test.dropna()
```

I.1.3. Limpieza y normalización del dataset

```
def limpiar(texto):
    # Se convierte todo el texto a minúsculas
    nuevo_texto = texto.lower()
    # Cambiar emojis por su significado textual
    nuevo_texto = emoji.demojize(nuevo_texto, language='es')
    # Eliminación de páginas web (palabras que empiezan por "http")
    nuevo_texto = re.sub('http\S+', ' ', nuevo_texto)
    # Eliminación de usuarios
    nuevo_texto = re.sub('user', ' ', nuevo_texto)
    # Eliminación de signos de puntuación
    regex = '[!"\#\$\%\&\'()*+,—./:;<=>?@\[\]^`{}~]'
    nuevo_texto = re.sub(regex , ' ', nuevo_texto)
    # Eliminación de números
    nuevo_texto = re.sub("\d+", ' ', nuevo_texto)
    # Eliminación de espacios en blanco múltiples
    nuevo_texto = re.sub("\\s+", ' ', nuevo_texto)
    return nuevo_texto
```

```
# Se aplica la función de limpieza a cada tweet
tweets_train['tweet_n']=tweets_train['tweet'].apply(lambda x: limpiar(x
    ))
tweets_dev['tweet_n']=tweets_dev['tweet'].apply(lambda x: limpiar(x))
tweets_test['tweet_n']=tweets_test['tweet'].apply(lambda x: limpiar(x))
tweets_train = [list(x) for x in tweets_train[['tweet_n', 'emotion']].
    values]
tweets_dev = [list(x) for x in tweets_dev[['tweet_n', 'emotion']].
    values]
tweets_test = [list(x) for x in tweets_test[['tweet_n']].values]
test_gold = test_gold.values.tolist()

def normalizar(tweets, min_words=2):
""" Función que dada una lista de tweets ([tweet, clase]), normaliza
    los tweets y devuelve una lista con los tweets normalizados,
    descartando aquellos tweets que tras la normalización tengan menos
    de "min_words" palabras en el tweet. """
    tweets_list = []
    for tweet in tqdm(tweets):
        # Tokenizamos el tweet realizando los puntos 1,2 y 3.
        tw=nlp(tweet[0].lower().replace('.',' ').replace('#',' ').strip
            ())
        # Normalizamos Puntos 4,5,6,7 y 8
        tw=([word.lemma_ for word in tw if (not word.is_punct)
                and (len(word.text) > 2) and (not word.is_stop)
                and (not word.text.startswith('https://'))
                and (not ':' in word.text)])
        # Eliminamos los tweets que tras la normalización tengan menos
            de  "min_words" palabras
        if len(tw) >= min_words:
            tweets_list.append([" ".join(tw), tweet[1]])
    return tweets_list
```

```
def normalizar_test(tweets, emotions, min_words=2):
""" Función que dada una lista de tweets ([tweet]) y una lista de
    etiquetas ([clase]), normaliza los tweets y devuelve ambas listas
    con los tweets normalizados, descartando aquellos tweets (y
    etiquetas respectivas) que tras la normalización tengan menos de "
    min_words" palabras en el tweet.
"""
    tweets_list = []
    emotions_list = []
    for i, tweet in tqdm(enumerate(tweets)):
        # Tokenizamos el tweet realizando los puntos 1,2 y 3.
        tw = nlp(tweet[0].lower().replace('.', ' ').replace('#', ' ').
            strip())
        # Normalizamos Puntos 4,5,6,7 y 8
        tw = ([word.lemma_ for word in tw if (not word.is_punct)
                and (len(word.text) > 2) and (not word.is_stop)
                #and (not word.text.startswith('@'))
                and (not word.text.startswith('https://'))
                and (not ':' in word.text)])
        # Eliminamos los tweets que tras la normalización tengan menos
            de "min_words" palabras
        if len(tw) >= min_words:
            tweets_list.append([" ".join(tw)])
            emotions_list.append(emotions[i][1])
    return tweets_list, emotions_list

# Normalizamos las frases
X_norm_train = normalizar(tweets_train)
X_norm_dev = normalizar(tweets_dev)
X_norm_test, y_test_gold = normalizar_test(tweets_test, test_gold)
```

```
#Guardado de los tweets normalizados en un fichero binario
filename = './models/normalized_train_tweets_string.pickle'
save_list = open(filename,"wb")
pickle.dump(X_norm_train, save_list)
save_list.close()
filename = './models/normalized_dev_tweets_string.pickle'
save_list = open(filename,"wb")
pickle.dump(X_norm_dev, save_list)
save_list.close()
filename = './models/normalized_test_tweets_string.pickle'
save_list = open(filename,"wb")
pickle.dump(X_norm_test, save_list)
save_list.close()
filename = './models/gold_test_string.pickle'
save_list = open(filename,"wb")
pickle.dump(y_test_gold, save_list)
save_list.close()

#Lectura de los tweets normalizados de un fichero binario
filename = './models/normalized_train_tweets_string.pickle'
X_norm_train = pickle.load(open(filename, 'rb'))
filename = './models/normalized_dev_tweets_string.pickle'
X_norm_dev = pickle.load(open(filename, 'rb'))
filename = './models/normalized_test_tweets_string.pickle'
X_norm_test = pickle.load(open(filename, 'rb'))
```

I.1.4. Extracción de características (Bolsa de Palabras)

```
#El particionado de datos ya está hecho
#pero hay que crear el diccionario de palabras

# Divido los datos en dos listas: X (los tweets) e y (polaridad)
```

```
X_train = [doc[0] for doc in X_norm_train]
y_train = np.array([doc[1] for doc in X_norm_train])
X_dev = [doc[0] for doc in X_norm_dev]
y_dev = np.array([doc[1] for doc in X_norm_dev])

# Pasamos los tweets normalizados a Bolsa de palabras
vectorizer = TfidfVectorizer(max_features=1500)
vectorizer.fit(X_train)
X_train = vectorizer.transform(X_train)
X_dev = vectorizer.transform(X_dev)
```

I.1.5. Creación de los modelos

```
mnb = MultinomialNB()
bnb = BernoulliNB()
svm_lin = SVC(kernel='linear')
svm_poly = SVC(kernel='poly')
svm_rbf = SVC(kernel='rbf')
svm_sig = SVC(kernel='sigmoid')
rf_20 = RandomForestClassifier(n_estimators=500, bootstrap=True,
    criterion='gini', max_depth=20, random_state=0)
rf_50 = RandomForestClassifier(n_estimators=500, bootstrap=True,
    criterion='gini', max_depth=50, random_state=0)
clasificadores = {'Multinomial NB': mnb,
                  'Bernoulli NB': bnb,
                  'SVM lineal': svm_lin,
                  'SVM polinomico': svm_poly,
                  'SVM Kernel rbf': svm_rbf,
                  'SVM Kernel Sigmoid': svm_sig,
                  'Random Forest d_20': rf_20,
                  'Random Forest d_50': rf_50}
```

```
# Ajustamos los modelos
for k, v in clasificadores.items():
    print ('CREANDO MODELO: {clas}'.format(clas=k))
    model = {}
    model['name'] = k
    # Entrenamiento
    v.fit(X_train, y_train)
```

I.1.6. Evaluación de los modelos

```
def evaluation(model, name, X_train, y_train, X_dev, y_dev):
""" Función de devuelve en un diccionario las métricas de evaluación de
    Accuracy, Precision, Recall y F1 para los conjuntos de datos de
   entrenamiento y validación
"""
    model_dict = {}
    model_dict['name'] = name
    y_pred_train = model.predict(X_train)
    y_pred_dev = model.predict(X_dev)
    model_dict['accuracy_train'] = accuracy_score(y_true=y_train,
       y_pred=y_pred_train)
    model_dict['accuracy_dev'] = accuracy_score(y_true=y_dev, y_pred=
       y_pred_dev)
    model_dict['precision_train'] = precision_score(y_true=y_train,
       y_pred=y_pred_train, average='weighted')
    model_dict['precision_dev'] = precision_score(y_true=y_dev, y_pred=
       y_pred_dev, average='weighted')
    model_dict['recall_train'] = recall_score(y_true=y_train, y_pred=
       y_pred_train, average='weighted')
    model_dict['recall_dev'] = recall_score(y_true=y_dev, y_pred=
       y_pred_dev, average='weighted')
```

```
    model_dict['f1_train'] = f1_score(y_true=y_train, y_pred=
        y_pred_train, average='weighted')
    model_dict['f1_dev'] = f1_score(y_true=y_dev, y_pred=y_pred_dev,
        average='weighted')
    return model_dict

# Calculamos las métricas de los modelos por separado
evaluacion = list()
for key, model in clasificadores.items():
    evaluacion.append(evaluation(model=model, name=key,
                                 X_train=X_train, y_train=y_train,
                                 X_dev=X_dev, y_dev=y_dev
                                 ))

# Pasamos los resultados a un DataFrame para visualizarlos mejor
df = pd.DataFrame.from_dict(evaluacion)
df.set_index("name", inplace=True)
df
```

I.1.7. Exportación de los mejores modelos y Bolsa de Palabras

```
# Creación de los modelos con todos los tweets
model_1 = SVC(kernel='sigmoid')
model_1.fit(X_train, y_train)
model_2 = SVC(kernel='linear')
model_2.fit(X_train, y_train)
model_3 = SVC(kernel='rbf')
model_3.fit(X_train, y_train)
model_4 = RandomForestClassifier(n_estimators=500, bootstrap=True,
    criterion='gini', max_depth=50, random_state=0)
model_4.fit(X_train, y_train)
```

```
#Obtenemos las métricas de los modelos con los datos de validación
labels = ['anger', 'disgust', 'fear', 'joy', 'others', 'sadness', '
    surprise']
print('Classification report model 1')
print(classification_report(y_true=y_dev, y_pred=model_1.predict(X_dev)
    , labels=labels))
print('Classification report model 2')
print(classification_report(y_true=y_dev, y_pred=model_2.predict(X_dev)
    ,labels=labels))
print('Classification report model 3')
print(classification_report(y_true=y_dev, y_pred=model_3.predict(X_dev)
    ,labels=labels))
print('Classification report model 4')
print(classification_report(y_true=y_dev, y_pred=model_4.predict(X_dev)
    ,labels=labels))
#Exportamos los modelos generados por el Algoritmo de Aprendizaje.
filename = './models/model_1_tweets.pickle'
save_model_1 = open(filename,"wb")
pickle.dump(model_1, save_model_1)
save_model_1.close()
filename = './models/model_2_tweets.pickle'
save_model_2 = open(filename,"wb")
pickle.dump(model_2, save_model_2)
save_model_2.close()
filename = './models/model_3_tweets.pickle'
save_model_3 = open(filename,"wb")
pickle.dump(model_3, save_model_3)
save_model_3.close()
filename = './models/model_4_tweets.pickle'
save_model_4 = open(filename,"wb")
pickle.dump(model_4, save_model_4)
save_model_4.close()
```

```
#Exportar la Bolsa de Palabras generada por la clase TfidfVectorizer.
filename = './models/vectorizer_bow_tweets.pickle'
save_bow = open(filename,"wb")
pickle.dump(vectorizer, save_bow)
save_bow.close()
```

I.1.8. Importación de los modelos y Bolsa de Palabras

```
#Importamos los modelos para la clasificación
filename = './models/model_1_tweets.pickle'
classifier_model_1 = pickle.load(open(filename, 'rb'))
filename = './models/model_2_tweets.pickle'
classifier_model_2 = pickle.load(open(filename, 'rb'))
filename = './models/model_3_tweets.pickle'
classifier_model_3 = pickle.load(open(filename, 'rb'))
filename = './models/model_4_tweets.pickle'
classifier_model_4 = pickle.load(open(filename, 'rb'))

#Importamos el modelo para crear la Bolsa de Palabras
filename = './models/vectorizer_bow_tweets.pickle'
classifier_model = pickle.load(open(filename, 'rb'))

#Creamos la Bolsa de Palabras
filename = './models/normalized_test_tweets_string.pickle'
X_norm_test = pickle.load(open(filename, 'rb'))
filename = './models/gold_test_string.pickle'
y_test = pickle.load(open(filename, 'rb'))
X_test = [doc[0] for doc in X_norm_test]
vectorizer = TfidfVectorizer(max_features=1500)
vectorizer.fit(X_test)
X_test = vectorizer.transform(X_test)
```

I.1.9. Predicción

```
predictions_1 = classifier_model_1.predict(X_test)
predictions_2 = classifier_model_2.predict(X_test)
predictions_3 = classifier_model_3.predict(X_test)
predictions_4 = classifier_model_4.predict(X_test)

#Obtenemos las métricas de los modelos con los datos de prueba
labels = ['anger','disgust','fear','joy','others','sadness','surprise']
print(classification_report(y_true=y_test, y_pred=predictions_1, labels
    =labels))
print(classification_report(y_true=y_test, y_pred=predictions_2, labels
    =labels))
print(classification_report(y_true=y_test, y_pred=predictions_3, labels
    =labels))
print(classification_report(y_true=y_test, y_pred=predictions_4, labels
    =labels))
```

I.2. Algoritmo Deep Learning

I.2.1. Librerías

```
import pandas as pd
import numpy as np
import torch
from transformers import pipeline
from transformers import AutoTokenizer,
    AutoModelForSequenceClassification, AutoConfig
from tqdm import tqdm
from sklearn.metrics import classification_report
```

I.2.2. Carga de datos

```
tweets_test = pd.read_csv('./data/emoevales_test.tsv', sep='\t+')
test_gold = pd.read_csv('./data/emoevales_test_gold.tsv', sep='\t+')
```

I.2.3. Carga del modelo XLM-RoBERTa

```
def preprocess(text):
    new_text = []
    for t in text.split(" "):
        t = '@user' if t.startswith('@') and len(t) > 1 else t
        t = 'http' if t.startswith('http') else t
        new_text.append(t)
    return " ".join(new_text)

model_path = "daveni/twitter-xlm-roberta-emotion-es"
tokenizer = AutoTokenizer.from_pretrained(model_path)
config = AutoConfig.from_pretrained(model_path)

model = AutoModelForSequenceClassification.from_pretrained(model_path)

emotion_analysis = pipeline("text-classification", framework="pt",
   model=model_path, tokenizer=model_path)
```

I.2.4. Predicción

```
predictions = []
for i in tqdm(range(len(tweets_test))):
    predictions.append(emotion_analysis(tweets_test.loc[i,'tweet'])[0][
        'label'])
labels = ['anger','disgust','fear','joy','others','sadness','surprise']
print(classification_report(y_true=test_gold['emotion'].values.tolist()
   , y_pred=predictions, labels=labels))
```