Análisis Automatizado de sentimientos en YouTube: Un enfoque basado en Machine Learning
Automated Sentiment Analysis on YouTube: A Machine Learning Approach

Trabajo de Fin de Grado
Curso 2023–2024

Autores
Jorge Villacorta
Alejandro Tabernero
Alonso Mata

Directores
Mercedes García Merayo
Manuel Méndez Hurtado

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Análisis Automatizado de sentimientos en YouTube: Un enfoque basado en Machine Learning
Automated Sentiment Analysis on YouTube: A Machine Learning Approach

Trabajo de Fin de Grado en Ingeniería Informática

Autores
Jorge Villacorta
Alejandro Tabernero
Alonso Mata

Directores
Mercedes García Merayo
Manuel Méndez Hurtado

Convocatoria: *Junio 2024*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Dedicatoria

A nuestra familia, por su apoyo incondicional
y no perder la fe en nosotros

Agradecimientos

A Manuel y Mercedes por su guía y ayuda durante todo el desarrollo del proyecto, a nuestros padres por todo el apoyo emocional proporcionado, a Nina por no desesperarse con los periodos de estrés, y a todos los que nos han ayudado a llegar hasta aquí.

Resumen

Análisis Automatizado de sentimientos en YouTube: Un enfoque basado en Machine Learning

El propósito fundamental de este Trabajo de Fin de Grado radica en examinar detenidamente los sentimientos de la población en torno a un tema polémico, haciendo uso de modelos y técnicas de machine learning con el fin de extraer información relevante y realizar un análisis exhaustivo de la opinión pública. Las redes sociales, como Twitter o YouTube, se presentan como fuentes de información excepcionalmente valiosas para captar la percepción sobre diversos temas, independientemente de su temática o naturaleza. Por lo tanto, los comentarios que conformarán nuestro corpus de análisis serán extraídos de una de estas dos plataformas, como detallaremos más adelante. Una vez obtenidos estos comentarios, se requerirá llevar a cabo un proceso de preprocesamiento que incluirá una limpieza de texto, eliminando elementos como emoticonos, marcas de tiempo y demás información no relevante. Posteriormente, se realizará un procesamiento de lenguaje natural con el objetivo de transformar los comentarios en datos que un modelo de aprendizaje automático pueda utilizar de manera efectiva. Una vez completado este preprocesamiento, se procederá a la selección de diversos modelos de aprendizaje y a su correspondiente entrenamiento. La elección de estos modelos es crucial, especialmente en el ámbito de análisis de sentimientos, donde algunos modelos son más apropiados para esta casuística que otros, como se detallará en los capítulos siguientes. En nuestro caso particular, hemos optado por emplear seis modelos distintos, los cuales analizaremos posteriormente. Esta cantidad nos proporciona una variedad suficiente para llevar a cabo un entrenamiento completo y contar con diversas opciones al momento de realizar nuestro análisis final. En la etapa final, llevaremos a cabo la evaluación de todos los modelos, comparando sus métricas y extrayendo atributos esenciales del conjunto de datos. Entre estos atributos se incluyen las palabras más relevantes para cada clase, el balance de clases en sí, así como métricas clave como precisión, exhaustividad y puntuación F1. Herramientas adicionales, como la matriz de confusión de cada modelo, servirán como referencia para seleccionar el modelo que mejor se ajuste a nuestro conjunto de datos, permitiéndonos realizar análisis de sentimientos futuros con un alto nivel de precisión.

Palabras clave

Análisis de Sentimientos, YouTube, Comentarios, Web Scraping, Procesamiento del Lenguaje Natural, NLP, Clasificación de Texto, Opinión Pública, Redes Sociales, Inteligencia Artificial, Python, Aprendizaje Automático, YouTube, OpenAI.

Abstract

Automated Sentiment Analysis on YouTube: A Machine Learning Approach

The fundamental purpose of this Bachelor's Thesis lies in thoroughly examining the sentiments of the population regarding a controversial topic, utilizing models and machine learning techniques to extract relevant information and conduct a comprehensive analysis of public opinion. Social media platforms, such as Twitter or YouTube, emerge as exceptionally valuable sources of information to grasp perceptions on various subjects, regardless of their theme or nature. Therefore, the comments that will constitute our analytical corpus will be extracted from one of these two platforms, as detailed later on. Once these comments are obtained, it will be necessary to undergo a preprocessing process, including text cleaning by eliminating elements such as emoticons, timestamps, and other non-relevant information. Subsequently, a natural language processing will be performed with the aim of transforming the comments into data that a machine learning model can effectively utilize. Upon completion of this preprocessing, the selection of various learning models and their respective training will take place. The choice of these models is crucial, especially in the field of sentiment analysis, where some models are more suitable for this scenario than others, as detailed in the following chapters. In our particular case, we have opted to employ six different models, which we will analyze later on. This quantity provides us with sufficient variety to carry out comprehensive training and have different options when conducting our final analysis. In the concluding stage, we will undertake the evaluation of all models, comparing their metrics and extracting essential attributes from the dataset. Among these attributes are the most relevant words for each class, the class balance itself, as well as key metrics such as precision, recall, and F1 score. Additional tools, such as the confusion matrix for each model, will serve as a reference to select the model that best fits our dataset, allowing us to conduct future sentiment analyses with a high level of precision.

Keywords

Sentiment Analysis, YouTube, Comments, Web Scraping, Natural Language Processing, Text Classification, Public Opinion, Social Media, Artificial Intelligence.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"Nuestra inteligencia es lo que nos hace humanos, y la IA es una extensión de esa cualidad"
— Yann LeCun

Estamos en la que probablemente sea la época de mayor conexión interpersonal de la historia de la humanidad, entre el siglo XX y XXI se han conseguido grandes hitos culturales, la invención de la radio, el transistor, el microprocesador o Internet. Este último invento ha condicionado el presente y futuro de nuestra especie de una forma que nadie pudo imaginar, por ejemplo, nadie se imagina ya un mundo en el que no existan aplicaciones como Google Maps para escoger al instante la mejor ruta en una ciudad en la que nunca habíamos estado. Mucha gente ni siquiera se informa a través de medios convencionales como la televisión o el periódico, consiguen toda la información a través de medios digitales o redes sociales. Hay redes sociales enteramente dedicadas a expresar opiniones como por ejemplo Twitter, o incluso Youtube es habitáculo de medios de comunicación independientes, influencers y demás habladores.

El tema elegido para realizar el estudio es Open AI, buscamos comprender cómo la gente percibe y reacciona ante la compañía y cómo sus emociones y sentimientos evolucionan con el tiempo. Para lograrlo, recopilaremos una gran cantidad de datos utilizando técnicas de *web scraping* y los clasificaremos en función del sentimiento que transmiten. Posteriormente, usaremos diferentes modelos de clasificación para probar y validar nuestro análisis con los datos obtenidos.

Aunque este enfoque se centra específicamente en comentarios de YouTube, creemos que proporcionará percepciones valiosas sobre los sentimientos en torno a Open AI. Estamos seguros de que los datos obtenidos de estas fuentes ofrecerán una comprensión integral de la opinión pública.

En resumen, nuestro estudio contribuirá al creciente cuerpo de investigación sobre el análisis de sentimientos de datos en redes sociales y proporcionará información sobre las emociones y opiniones del público relacionadas con Open AI.

1.1. Motivación

Open AI ha pasado del absoluto desconocimiento por parte del gran público a ser uno de los temas más hablados en todo tipo de tertulias y debates, tanto a nivel académico como en círculos más íntimos. Las discusiones en torno a esta compañía y sus productos, siendo ChatGPT el más relevante de todos, abarcan desde el futuro de la humanidad hasta tesis que proponen reestructurar los métodos de aprendizaje en todas las etapas del ciclo educativo. Debido al interés suscitado por Open AI en todas las esferas de las sociedad, es un tema ideal para realizar un estudio para tratar de encontrar y analizar emociones y sentimientos expresados por los usuarios en comentarios de vídeos de Youtube relacionados con Open AI aplicando técnicas sofisticadas de procesamiento del lenguaje natural.

1.2. Objetivos

El fin principal de esta investigación es evaluar las emociones y puntos de vista que los usuarios expresan en YouTube sobre el impacto de las IAs en nuestra sociedad, más concretamente que impacto tiene la empresa OpenAI, desarrolladora de la IA más popular a día de hoy, ChatGPT. Mediante este análisis detallado, pretendemos determinar la opinión de la sociedad respecto a este tema, además de obtener una visión más profunda de las características de las diferentes opiniones y ayudarnos a comprender en profundidad como funcionan estos modelos y que ventajas tienen entre ellos.

1.3. Plan de trabajo

Este proyecto se ha dividido en cinco grandes etapas, las cuales eran dependientes unas de otras. Esta dependencia ha provocado que se hayan ido realizando de manera secuencial, a excepción del desarrollo de esta memoria que se realizó de manera concurrente al resto de etapas.

- **Estudio inicial:** En esta etapa el equipo se familiarizó con el proyecto, haciendo lecturas sobre análisis de sentimientos en redes sociales, diferentes tipos de modelos que se pueden emplear e investigando que temas nos podían ser de utilidad. Esta última parte fue algo problemática más adelante ya que se cambió de tema debido a dificultades para procesar y analizar textos en español ya que la gran mayoría de herramientas están preparadas para textos en inglés.

- **Selección de Herramientas:** Tras el estudio inicial, el equipo se hizo una idea de como enfocar el proyecto y en esta nueva fase se estudiaron los diferentes tipos de herramientas que se podían usar para llevarlo acabo. Tras muchas pruebas con diferentes tipos de herramientas para Twitter, desde Twint hasta Tweepy se desistió, ya que con los cambios en la plataforma el acceso a la API era de pago y este hecho está en contra de los principios morales del

equipo. Se seleccionó Youtube como red social para realizar el estudio debido a la apertura de su API y la facilidad para extracción de datos. Tras probar varias herramientas como Octoparse, nos decantamos por un acceso directo a la API de Youtube a través de scripts de Python. Depués investigamos sobre herramientas como TextBlob o Stanza, para realizar el análisis de los sentimientos y etiquetar los atributos, pero finalmente decidimos usar NLTK debido a su sencillez y transparencia.

- **Preprocesamiento:** Esta etapa, que inicialmente no se consideró que fuera a ser demasiado laboriosa, resultó ser una de las más complejas y largas en el tiempo en términos de desarrollo. Primero realizamos la selección de todos los vídeos del tema original en español, y una vez obtuvimos todos los datos de nuestro dataset descubrimos que la mayoría de las herramientas de análisis de sentimientos sólo funcionaban con texto en inglés, lo que provocó que realizáramos el cambio de tema a algo más internacional. Tras esto la elaboración de los *scripts* requirió de tareas de estudio de Python para poder elaborarlos de la mejor manera posible.

- **Entrenamiento:** En esta fase del proyecto, hicimos uso de los datos preprocesados para entrenar los modelos. Tras el estudio inicial teníamos una idea de los modelos más populares empleados en el ámbito del análisis de sentimiento, pero en esta fase se realizó un estudio más en profundidad de dichos modelos. Se probaron varios modelos y se intentó mejorar todo lo posible los resultados obtenidos, hasta seleccionar los más relevantes para este trabajo.

- **Evaluación de los resultados:** Tras entrenar los modelos de la forma más óptima posible, llegó el momento de estudiar los resultados obtenidos para sacar las conclusiones del proyecto. Se diseñaron *scripts* para crear gráficas con datos que el equipo encontró relevantes y representativas.

- **Documentación:** Esta etapa se corresponde con el desarrollo de la memoria del proyecto. Se intentó, dentro de lo posible, que fuera a la par con las tareas de desarrollo del proyecto.

En la figura 1.1 podemos apreciar de manera gráfica el desarrollo del proyecto.

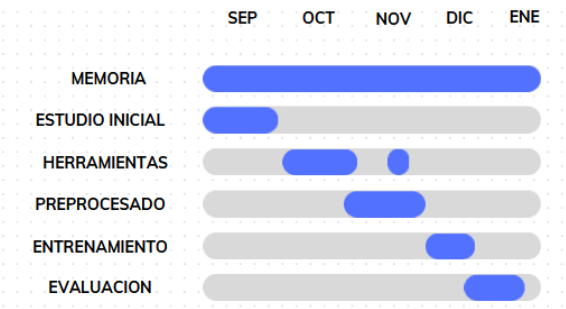

Figura 1.1: Diagrama de Gantt

Capítulo 2

Recursos Tecnológicos

Antes de empezar con el estudio en sí, veremos las herramientas utilizadas para llevarlo a cabo. La intención de este capítulo es describir tanto la instalación como el uso del entorno de trabajo elegido para el proyecto indicando la finalidad de cada herramienta y el papel que ocupan en el entorno de trabajo.

2.1. Anaconda

Anaconda es una distribución de Python principalmente útil en la ciencia de datos, el aprendizaje automático y la computación científica. Su uso busca simplificar el manejo de paquetes y entornos, y es fundamental por varias razones:

- **Gestión de Entornos**: Anaconda permite crear y manejar entornos aislados, lo cual es muy útil para mantener la coherencia y reproducibilidad del proyecto. Permite dar a cada entorno su propia configuración y conjunto de bibliotecas.
- **Facilitación de Instalaciones Complejas**: Muchas de las bibliotecas utilizadas en este proyecto, como PyTorch y TensorFlow, tienen múltiples dependencias que pueden ser difíciles de configurar. Anaconda simplifica este proceso, gestionando automáticamente estas dependencias.
- **Amplia Compatibilidad**: Esta distribución es compatible con múltiples sistemas operativos, lo que asegura que el proyecto pueda ser replicado y ejecutado en diferentes plataformas sin problemas.
- **Ecosistema Rico**: Anaconda ofrece acceso a una vasta colección de bibliotecas y herramientas para ciencia de datos y aprendizaje automático, lo que ha sido crucial para abordar las diversas necesidades del proyecto, desde el procesamiento de datos hasta la modelización avanzada.

Instalación y Preparación de Anaconda

- Descargar el Instalador: Visitar el sitio web de Anaconda y descargar el instalador adecuado para el sistema operativo.

- Ejecutar el Instalador: Seguir las instrucciones de instalación proporcionadas por el instalador.
- Verificar la Instalación: Abrir la terminal o Anaconda Prompt y ejecutar 'conda list' para asegurarse de que Conda esté instalado correctamente.
- Crear un Entorno: Crearemos un entorno específico para el proyecto con el comando

```
$ conda create --mi_entorno python=3.8
```

- Activar el Entorno: Activar el entorno creado con el comando antes de comenzar a trabajar en el proyecto.

```
$ conda activate mi_entorno
```

2.2. PyTorch

PyTorch es un marco de trabajo de código abierto para aprendizaje profundo, reconocido por su flexibilidad y enfoque orientado a la investigación. Facilita el cálculo tensorial y la creación de redes neuronales, siendo esencial para la carga y operación de modelos de NLP en nuestro proyecto.

Instalación y Preparación

La instalación se realiza a través de Conda o pip. Dependiendo de la configuración del sistema y si se utiliza CUDA para GPU, el comando de instalación puede variar. Ejemplo de comandos:

```
$ conda install pytorch torchvision torchaudio -c pytorch
```

```
$ conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
```

2.3. Transformers (Hugging Face)

Transformers de Hugging Face ofrece una amplia gama de modelos de lenguaje natural pre-entrenados, utilizados en nuestro estudio para acceder a modelos como GPT-2 y facilitar tareas de NLP.

Instalación y Preparación

Se instala mediante pip, siendo necesario tener PyTorch o TensorFlow previamente instalados:

```
$ pip install transformers
```

2.4. Pandas

Pandas es una biblioteca de Python para análisis y manipulación de datos, usada en nuestro proyecto para manejar conjuntos de datos, incluyendo la lectura y transformación de comentarios de YouTube.

Instalación y Preparación

Pandas se instala con pip o conda:

```
$ pip install pandas
```

```
$ conda install pandas
```

2.5. Selenium

Selenium automatiza navegadores web y es usado en nuestro proyecto para el scraping de comentarios de YouTube, imitando la navegación e interacción de usuarios.

Instalación y Preparación

Selenium se instala a través de pip. Además, se requiere el driver del navegador a automatizar:

```
$ pip install selenium
```

2.6. Jupyter Notebook

Jupyter Notebook es una interfaz interactiva que permite el desarrollo de código dinámico con Python. Los usuarios pueden ejecutar celdas de código de manera interactiva, ver resultados y combinar código con texto explicativo, lo que lo convierte en una herramienta poderosa para análisis de datos, aprendizaje automático e investigación colaborativa. Proporciona un entorno interactivo y flexible para escribir, probar y documentar código y flujos de trabajo de análisis.

Instalación y Preparación

Instalamos Jupyter en nuestro entorno conda con el comando:

```
$ pip install jupyter
```

A continuación, abrimos nuestra interfaz de Jupyter Notebook ejecutando en consola:

```
$ jupyter notebook
```

Este comando abrirá una pestaña en su navegador web predeterminado. Esta pestaña mostrará una lista de cuadernos, archivos y directorios dentro de la carpeta donde iniciamos Jupyter, como se muestra en la Figura.

2.7. Notion

Notion es una herramienta versátil con la capacidad de centralizar tareas y flujos de trabajo. Funciona como una plataforma de colaboración y gestión de proyectos, permitiendo la creación de documentos, bases de datos y tableros Kanban en un entorno digital intuitivo. Su diseño modular y personalizable se adapta a diversas necesidades, facilitando la planificación, la toma de notas y la gestión de equipos.

Instalación y Preparación

Notion es una aplicación que puede usarse descargada tanto como en una versión web.

Capítulo 3

Extracción de los Datos

La extracción y preprocesado de datos constituyen uno de los pasos más críticos de este proyecto y sirven como fundamento para el futuro entrenamiento de los modelos. La extracción de datos se reveló como un desafío debido a varias razones. En primer lugar, se presentó la dificultad de seleccionar la red social de la cual deseamos extraer la información. Como se indicó anteriormente, nuestras opciones principales eran YouTube y Twitter. No obstante, debido a recientes cambios en la propiedad y dirección ejecutiva de Twitter, su API se encuentra desactualizada y no resulta utilizable. Debido a esto, la opción de emplear Twitter/X fue descartada, debido a su alto coste, a pesar de ser una fuente muy rica de opiniones sobre diversos temas. En última instancia, todo nuestro conjunto de datos será extraído exclusivamente de YouTube, ya que también consideramos que esta aplicación puede aportarnos información muy valiosa.

Nuestra siguiente tarea se centró en crear una herramienta que nos facilitará la extracción de los datos que queremos utilizar para entrenar nuestros modelos. Inicialmente, consideramos que la opción más conveniente sería utilizar un *software* de extracción disponible en línea, con la expectativa de ahorrar tiempo de manera significativa. Sin embargo, notamos que la mayoría de estas herramientas requieren una suscripción "*premium*" o eran demasiado potentes para la sencilla tarea que queríamos realizar. Finalmente, reconsideramos nuestra estrategia de extracción de datos y concluimos que nuestro caso específico no requería una solución excesivamente compleja, ya que solo necesitábamos los comentarios de cada vídeo y, como máximo, la identificación del usuario que los escribió. Después de analizar esta situación, decidimos desarrollar un *script* en Python que, mediante el uso de la API de YouTube, extrae los comentarios de cada vídeo proporcionado a través de un archivo de texto que contiene los identificadores correspondientes. Este *script* inicialmente se creó para extraer comentarios, pero con el tiempo evolucionó hasta convertirse en una herramienta que realiza tanto la extracción como el preprocesado del conjunto de datos completo. Consideramos que esta aproximación a la extracción y preprocesado es mucho mas interesante, ya que se puede ver todo el proceso desde cero y se pueden observar que herramientas se han utilizado en cada paso, aparte de añadirle una capa más de profundidad a nuestro trabajo. Las distintas etapas de este proceso se detallan a continuación.

Modulo	Librería	Descripción
os	os	Funcionalidades dependientes del sistema operativo.
subprocess	subprocess	permite lanzar nuevos procesos y conectarse a sus pipes de entrada/salida/error.
sys	sys	Funciones y variables que interactúan con el sistema operativo.

Tabla 3.1: Módulos del *script* principal

3.1. *script* principal

El *script* principal, que aparece en la Figura 3.1, se encarga de administrar un archivo CSV proporcionado como argumento del *script*, así como el archivo de texto mencionado anteriormente. Este *script* lleva a cabo diversas modificaciones en el archivo CSV de destino. En resumen, coordina y ejecuta todas las fases de extracción y preprocesado de los comentarios. En una primera fase, se eliminan todos los archivos CSV existentes previamente, en caso de que no sea la primera vez que se ejecuta el *script* y se necesite eliminar archivos no relevantes. Luego, en un bucle, se extraen los comentarios de cada vídeo, los cuales se combinan posteriormente en un único archivo CSV destino, aquel que ha sido suministrado como argumento. Una vez extraídos los comentarios y combinados en nuestro archivo CSV de destino, se da por finalizada la extracción y empieza la etapa puramente de preprocesado, con pasos como la limpieza del texto, detección de sentimiento e idioma, tokenización y lematización. Al finalizar la ejecución de este *script*, tendremos nuestro conjunto de datos preparado para el entrenamiento de nuestros modelos de aprendizaje automático. En la Tabla 3.1 se pueden observar los módulos utilizados un poco más de tallados, que son *os* [4], *subprocess* [5], y *sys*.

3.2. Extracción de comentarios

Este *script*, ilustrado en la Figura 3.2, toma como argumento el ID del vídeo del cual se desean extraer los comentarios. Posteriormente, procede a extraer los comentarios usando la API y de YouTube y el modulo *requests* de Python [6], haciendo *scroll* en los comentarios del vídeo para obtener todos los comentarios disponibles (sin contar las respuestas a los comentarios), gracias al soporte HTML que ofrece *requests* [7]. Entre la información extraída, se incluye el nombre del usuario que escribió cada comentario, el número de *likes* que tiene dicho comentario, y el comentario en sí. Esta información se guarda en un archivo CSV con el formato `videoID_comments.csv`. El *script* se ejecuta de manera iterativa según la cantidad de identificadores de vídeo proporcionados en un archivo de texto específico. En la Tabla 3.2 se pueden observar los módulos utilizados.

```
import subprocess
import sys
import os

#destFile: nombre del fichero donde queremos guardar los comentarios
#idSourceFile: fichero desde el que extraremos los identificadores de cada video
destFile = sys.argv[1]
idSourceFile = sys.argv[2]

directory = os.getcwd()
#-----------------------------------------------
#Borramos todos los archivos .csv existentes
all_files = os.listdir(directory)
csv_files = [file for file in all_files if file.endswith(".csv")]
for csv_file in csv_files:
    os.remove(os.path.join(directory, csv_file))
    print(f'Deleted: {csv_file}')
#-----------------------------------------------
# Leemos cada ID del video y extraemos la informacion de este
with open(idSourceFile, 'r') as file:
    video_ids = [line.strip() for line in file]
for video_id in video_ids:
    # Utilizamos el script yt_scrapper para extraer los comentarios
    scrape_command = ["python", "yt_scrapper.py", video_id]
    subprocess.run(scrape_command)

print("Todos los comentarios han sido extraidos")
#-----------------------------------------------
#Combinamos todos los csv de cada video en uno unico
merge_command = ["python", "csv_merger.py", destFile]
subprocess.run(merge_command)
#Limpiamos y formateamos el csv de comentarios
clean_command = ["python", "pre_text.py", destFile]
subprocess.run(clean_command)
#Detectamos el idioma de cada comentario y eliminamos los que no son en ingles
detect_command = ["python", "lang_detect.py", destFile]
subprocess.run(detect_command)
#Analizamos el sentimiento de cada comentario
sentiment_command = ["python", "sentiment.py", destfile]
subprocess.run(sentiment_command)
#Tokenizamos y lematizamos los comentarios
tokenization_command = ["python", "tokenization.py", destfile]
subprocess.run(tokenization_command)
```

Figura 3.1: *script* Principal

Modulo	**Librería**	**Descripción**
pandas	pandas	Manipulación y análisis de datos de manera eficiente y fácil.
requests	requests	Facilita el envío de solicitudes HTTP y la gestión de aplicaciones web.
sys	sys	Funciones y variables que interactúan con el sistema operativo.

Tabla 3.2: Módulos del *script* de extracción de comentarios con la API de YouTube

3.3. Combinación de archivos CSV

Este *script* sencillo, como se puede apreciar en la Figura 3.3, tiene la función de combinar todos los archivos CSV generados por el *script* anterior, correspondientes a cada vídeo. Los fusiona en un solo archivo CSV, cuyo nombre se proporciona como argumento. La importancia de este *script* radica en su capacidad para simplificar el trabajo, permitiéndonos manejar todos nuestros datos de manera eficiente en un solo archivo. A partir de este paso, todas las fases de preprocesado se ejecutarán en este archivo objetivo. En la Tabla 3.3 se pueden observar los módulos utilizados.

3.4. Formateo de Texto

Después de extraer y consolidar todos los datos en un único archivo, procedemos con la limpieza de esta información. Dado que nuestro enfoque se centra exclusi-

```python
import pandas as pd
import requests
import sys

api_key = 'AIzaSyCut7GWauHbjoyOGxYY3TBMzTyjW1TSwdw'

#Identificador del video
video_id = sys.argv[1]

#YouTube Data API endpoint
api_endpoint = f'https://www.googleapis.com/youtube/v3/commentThreads'

#Nuestros datos de comentarios incialmente contendran el nombre de usuario, el comentario en si y el numero de likes
comment_data = {
    'User': [],
    'Comment': [],
    'Likes': [],
}

#Parametros para la API
params = {
    'part': 'snippet,replies',
    'videoId': video_id,
    'key': api_key,
    'maxResults': 3000,
}

next_page_token = None

while True:
    if next_page_token:
        params['pageToken'] = next_page_token

    response = requests.get(api_endpoint, params=params)
    data = response.json()

    for item in data['items']:
        snippet = item['snippet']['topLevelComment']['snippet']
        comment_data['User'].append(snippet['authorDisplayName'])
        comment_data['Comment'].append(snippet['textDisplay'])
        comment_data['Likes'].append(snippet['likeCount'])

    next_page_token = data.get('nextPageToken')

    if not next_page_token:
        break

# Creamos el DataFrame
df = pd.DataFrame(comment_data)

#Guardamos el dataframe en un CSV. Como nuestros comentarios pueden tener comas, nuestro separador sera |
csv_filename = f'{video_id}_comments.csv'
df.to_csv(csv_filename, sep = '|', index=False)

print(f"Datos guardados en {csv_filename}")
```

Figura 3.2: Extracción de comentarios

Modulo	**Librería**	**Descripción**
os	os	Funcionalidades dependientes del sistema operativo.
pandas	pandas	Manipulación y análisis de datos de manera eficiente y fácil.
sys	sys	Funciones y variables que interactúan con el sistema operativo.

Tabla 3.3: Módulos del *script* de combinación de ficheros .csv

```python
import os
import sys
import pandas as pd


directory = os.getcwd()

#Listamos todos los csv en el directorio actual
csv_files = [file for file in os.listdir(directory) if file.endswith(".csv")]

# Lista vacia en la que iremos almacenando los diferentes datframes
data_frames = []

# Leemos cada csv y concatenamos el dataframe a nuestra lista
for csv_file in csv_files:
    csv_file_path = os.path.join(directory, csv_file)
    df = pd.read_csv(csv_file_path, sep = '|')
    data_frames.append(df)

# Creamos un nuevo Dataframe con nuestra lista de datframes
merged_df = pd.concat(data_frames, ignore_index=True)

# Guardamos nuestro nuevo dataframe en un csv con el nombre del archivo destino
merged_csv_filename = sys.argv[1]
merged_df.to_csv(merged_csv_filename, index=False, sep ='|')

print(f'Todos los comentarios han sido fusionados en {merged_csv_filename}')

# Borramos todos los csv individuales excepto el que hemos usado para fusionar nuestros comentarios
for csv_file in csv_files:
    if csv_file != merged_csv_filename:
        os.remove(os.path.join(directory, csv_file))

print(f'Archivos individuales eliminados (excepto {merged_csv_filename}).')
```

Figura 3.3: Combinación de archivos CSV

Modulo	Librería	Descripción
html	html	Utilidades para modificar HTML.
pandas	pandas	Manipulación y análisis de datos.
re	re	Operaciones de coincidencia de expresiones regulares [9]
sys	sys	Funciones y variables que interactúan con el sistema operativo.

Tabla 3.4: Módulos del *script* de limpieza y formateo de los comentarios

vamente en los comentarios, la limpieza se realiza únicamente en la columna que contiene estos comentarios. Esta etapa se dedica principalmente a eliminar información no necesaria, como emoticonos, marcas HTML, *timestamps* y cualquier otro contenido del comentario que no sea una palabra válida para el futuro entrenamiento. Esto principalmente se puede realizar utilizando expresiones regulares [8]. Como resultado de esta operación, la columna de comentarios experimenta una modificación, almacenando ahora únicamente el texto plano del comentario. En la Figura 3.4 se ilustra el *script* empleado para este proceso. En la Tabla 3.4 se pueden observar los módulos utilizados.

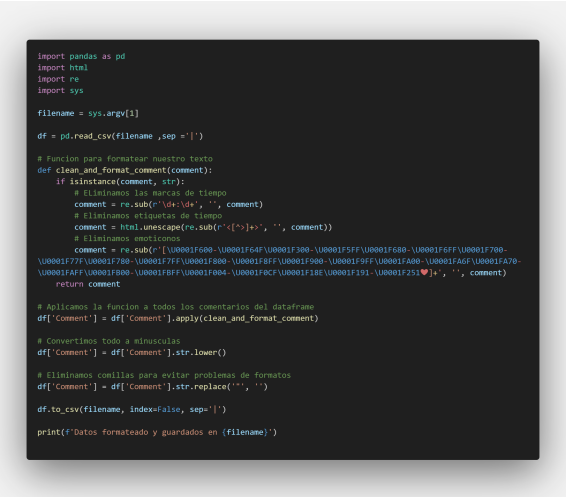

Figura 3.4: Formateo de Texto

3.5. Detección de Idioma

Al trabajar con comentarios de YouTube, nos enfrentamos a la posibilidad de que estos comentarios estén redactados en diversos idiomas, lo que podría complicar el preprocesado del texto. La presencia de varios idiomas en el conjunto de datos podría aumentar considerablemente la complejidad computacional de todo el *script*. Por esta razón, hemos tomado la decisión de unificar todos los comentarios en un único idioma empleando el *script* representado en la Figura 3.5, gracias al modulo

Modulo	Librería	Descripción
langdetect	langdetect	Detección de idioma
os	os	Funcionalidades dependientes del sistema operativo.
pandas	pandas	Manipulación y análisis de datos.
sys	sys	Funciones y variables que interactúan con el sistema operativo.

Tabla 3.5: Módulos del *script* de detección de idioma

lang-detect [10]. Dado que la temática y la mayoría de los vídeos se encuentran en inglés, hemos optado por que todos los comentarios en nuestro corpus estén en este idioma. Los comentarios que no estén en inglés son eliminados del corpus. Aunque esta decisión podría resultar en la pérdida de información, el porcentaje de comentarios en otros idiomas es muy pequeño, lo que no representa un riesgo o inconveniente significativo para el futuro entrenamiento de datos. Otra ventaja de realizar el entrenamiento en ingles, es que existen muchas más herramientas actualizadas en ingles que en español o en otro idioma, con lo cual disponemos de más posibilidades para realizar un entrenamiento realmente personalizado. En la Tabla 3.5 podemos observar los módulos utilizados en este *script*.

```
import os
import sys
import pandas as pd
from langdetect import detect

filename = sys.argv[1]

comments_df = pd.read_csv(filename, sep='|')

# Function para detectar el idioma usando el modulo langdetect
def detect_language(comment):
    try:
        return detect(comment)
    except:
        return 'unknown'

# Aplicamos la funcion de detección de idioma a cada comentario y creamos una columna nueva con el idioma
comments_df['Language'] = comments_df['Comment'].apply(detect_language)

# Filtramos y eliminamos los comentarios que no estan en ingles
en_comments_df = comments_df[comments_df['Language'] == 'en']

# Creamos una copia del DataFrame antes de realizar la asignación
en_comments_df_copy = en_comments_df.copy()

# Realizamos la asignación en la copia
en_comments_df_copy['ID'] = en_comments_df_copy.index

# Reorganizamos las columnas para que 'ID' esté al principio
en_comments_df_copy = en_comments_df_copy[['ID'] + [col for col in en_comments_df_copy.columns if col !=
'ID']]

en_comments_df_copy.to_csv(filename, sep='|', index=False)

print(f'Lenguaje detectado')
```

Figura 3.5: Detección de Idioma

3.6. Análisis de Sentimiento

Después de estandarizar el idioma del corpus, avanzaremos con el análisis de sentimiento con el propósito de obtener una variable objetivo que sea útil para el posterior entrenamiento de los datos. Esta variable objetivo se representa mediante una columna de clases multinominales, lo que implica que un comentario puede ser

Modulo	Librería	Descripción
nltk	nltk	Conjunto de bibliotecas y programas para el procesamiento del lenguaje natural.
pandas	pandas	Manipulación y análisis de datos.
sys	sys	Funciones y variables que interactúan con el sistema operativo.

Tabla 3.6: Módulos del *script* de detección de sentimiento

clasificado en más de dos categorías. En este contexto, un comentario puede ser etiquetado como "Positivo", "Negativo" o "Neutral". Este *script* resulta fundamental, ya que los comentarios en sí no poseen un atributo que indique su sentimiento, por lo que es necesario agregarlo para que los modelos de aprendizaje automático dispongan de una variable/atributo objetivo en el que basarse.

El análisis de sentimiento es crucial en la minería de texto, ya que permite extraer información valiosa sobre la actitud de los usuarios hacia determinados temas o productos. Con la incorporación de esta variable objetivo, nuestros datos están preparados para ser utilizados en el entrenamiento y evaluación de modelos de aprendizaje automático, los cuales podrán aprender patrones y relaciones entre el texto de los comentarios y sus respectivos sentimientos, mejorando así su capacidad de predicción en futuras instancias.

Para el análisis de sentimiento, hemos utilizado NLTK (Natural Language Toolkit), una biblioteca de procesamiento de lenguaje natural en Python [11]. NLTK proporciona herramientas y recursos para trabajar con texto, incluyendo funcionalidades específicas para el análisis de sentimiento [12]. Al emplear esta biblioteca, hemos llevado a cabo tareas como tokenización, lematización e incluso vectorización, aunque esta ultima la dejamos a cargo de cada modelo. Todo el proceso se realiza en el *script* ilustrado en la Figura 3.6. En la Tabla 3.6 podemos observar los módulos utilizados en este *script*.

Figura 3.6: Análisis de Sentimiento

3.7. Tokenización y lematización

Una vez determinado el sentimiento de cada comentario, que quizás sea el paso más relevante en todo el proceso de extracción de datos, ya que sin el sentimiento nuestros modelos no tienen una variable objetivo sobre la que entrenar, realizaremos la tokenización y lematización de cada comentario, haciendo uso del *script* representado en la Figura 3.7, en el que el modulo principal que se utiliza es Stanza, que es una colección de herramientas precisas y eficientes para el análisis lingüístico. [13]. Estos pasos nos ayudarán a presentar la información de manera simple y desglosada, permitiendo que nuestros modelos la interpreten más rápidamente y de una forma mucho más clara.

La tokenización, específicamente, consiste en subdividir el texto en unidades más pequeñas llamadas tokens, facilitando así su análisis y comprensión por parte de los algoritmos. Básicamente, tokenizar es dividir el comentario en palabras. Pero una vez tenemos el comentario tokenizado en palabras, necesitamos reducir esas palabras a su forma básica, ya sean verbos, sustantivos, adjetivos, etc. Es aquí donde entra la lematización.

Stemming vs lematización

Ambas son técnicas usadas en el procesado de lenguaje natural que reduce las palabras de entrada a su raíz o forma base, lo que facilita la interpretación y comparación de diferentes formas gramaticales de una misma palabra. Este paso contribuye a simplificar el conjunto de datos y a mejorar la eficacia de los modelos en el análisis de patrones lingüísticos, pero tienen una serie de diferencias [14].

El *stemming* [15] elimina los afijos de las palabras (sufijos, prefijos), dejando sólo la raíz o *stem*. El problema de este método es que puede resultar en palabras no válidas, por ejemplo, al aplicar el *stemming* a las palabras “pensando”, ”pensado” y “pensamiento” la raiz resultante sería “pensa” lo cual no es una palabra existente en castellano. Es por esto que optamos por la lematización, que aun siendo más compleja nos ofrece mejores resultados.

La lematización [16] es el proceso de reducir las palabras a su forma base o lema, siendo este una palabra con significado. Es un proceso más complejo que el *stemming* ya que no sirve con eliminar los afijos, sino que hay que analizar la palabra morfológicamente, sintácticamente y contextualmente. Además es común que se haga uso de una base de datos para conseguir estos objetivos.

Por ejemplo, consideremos las palabras **corriendo** y **corre**. Aplicando la lematización, ambas se reducirían al lema **correr**. Del mismo modo, las formas verbales **hablo** y **hablas** se lematizarían ambas como **hablar**. Este proceso permite que los modelos se centren en el significado de las palabras o tokens, ignorando las variaciones gramaticales [17]. En la Tabla 3.7 podemos observar los módulos utilizados en este *script*.

Una vez finalizado este paso, ya podríamos empezar a realizar el entrenamiento de los modelos. En un principio pensamos en incluir la vectorización en el preprocesado, pero nos dimos cuenta que para distintos modelos de aprendizaje automático, podemos encontrar diferentes vectorizaciones que se ajusten a estos y funcionen

```
import pandas as pd
import stanza
import sys
import nltk
from nltk.corpus import stopwords

#Usamos stanza para la tokenizacion
stanza.download('en')
nlp = stanza.Pipeline(lang='en', processors='tokenize,lemma')

# Definimos una funcion para realizar la tokenizacion y lematizacion
def tokenize(text):

    doc = nlp(text)

    # Extraemos los tokens
    tokens = [word.text for sent in doc.sentences for word in sent.tokens]
    return tokens

#funcion para eliminar los stop words de los comentarios, una vez tokenizado el comentario
def remove_stopwords(tokenized_text):
    nltk.download('stopwords')
    stop_words = set(stopwords.words('english'))
    filtered_text = [word for word in tokenized_text if word.lower() not in stop_words]
    return filtered_text


#funcion para lematizar los tokens
def lemmatize(tokenized_text):
    doc = nlp(' '.join(tokenized_text))  # Join tokens into a single string
    lemmas = [word.words[0].lemma if word.words else '' for sent in doc.sentences for word in sent.tokens]
    return lemmas


filename = sys.argv[1]
df = pd.read_csv(filename, sep='|')

comments = df['Comment']

# Aplicamos las funciones deifnidas previamente
results = comments.apply(tokenize).apply(remove_stopwords).apply(lemmatize)

# Creamos una nueva columna en la que guardar los lemas
df['TextData'] = results

df.to_csv(filename, sep='|', index=False)

print(f"Tokenizacion,eliminacion de stop words y lematizacion completadas. Resultados guardados en {
filename}")
```

Figura 3.7: Análisis de Sentimiento

mejor, como veremos en el siguiente capítulo.

Modulo	**Librería**	**Descripción**
nltk	nltk	Conjunto de bibliotecas y programas para el procesamiento del lenguaje natural.
pandas	pandas	Manipulación y análisis de datos.
stanza	stanza	Stanza es una colección de herramientas precisas y eficientes para el análisis lingüístico.
sys	sys	Funciones y variables que interactúan con el sistema operativo.
nltk.corpus	nltk	NLTK corpus nos da acceso directo a varias colecciones de textos.

Tabla 3.7: Módulos del *script* de tokenización y lematización

Capítulo 4

Modelos

En el presente capítulo abordaremos la implementación de modelos de clasificación automática de sentimientos aplicados a comentarios de la plataforma YouTube, como parte del campo de estudio del Procesamiento de Lenguaje Natural (NLP). La naturaleza textual y la riqueza expresiva de los comentarios hacen de esta fuente un corpus de estudio ideal para el entrenamiento y la evaluación de modelos de aprendizaje automático.

Dentro del aprendizaje automático, se distinguen dos categorías principales: el aprendizaje supervisado y el no supervisado. El primero se basa en conjuntos de datos donde cada entrada está asociada a una etiqueta o clase específica, facilitando la tarea de predecir la clasificación de nuevos datos. El segundo, por otro lado, intenta inferir la estructura de los datos sin etiquetas previas, identificando patrones y agrupaciones intrínsecas.

Para el propósito de esta tesis, hemos decidido emplear el aprendizaje supervisado puesto que trabajamos con un conjunto de datos ya etiquetados como vimos en el capítulo interior. Aunque estos etiquetados no han sido corroborados manualmente, creemos que representan un punto de partida válido para el desarrollo de los modelos, dado el volumen y la variabilidad de los datos con los que se trabaja.

A través de un proceso iterativo de entrenamiento, validación y ajuste de hiperparámetros, buscaremos optimizar la precisión y efectividad de nuestros modelos, con el fin de lograr un análisis de sentimientos lo más coherente y confiable posible.

En las siguientes secciones se procederá a detallar los modelos de clasificación y regresión seleccionados, destacando su adecuación al análisis de texto y su capacidad para manejar las dimensiones del conjunto de datos en cuestión. Se explicará cómo dichos modelos son entrenados para reconocer patrones en los comentarios y clasificarlos según su sentimiento.

Sin embargo, antes de poder entrenar los datos necesitamos vectorizarlos, ya que nuestros modelos de aprendizaje no pueden entrenar con texto como tal, si no que trabajan con datos numéricos, es por eso que necesitamos convertir nuestras palabras lematizadas previamente a vectores numéricos. No realizamos la vectorización durante el preprocesado ya que consideramos que podíamos utilizar distintas vectorizaciones según el modelo, dándonos mas opciones a la hora de optimizar nuestros modelos. En la siguiente sección se profundiza más en la vectorización y en las

diferentes opciones que decidimos utilizar.

4.1. Vectorización

Durante el entrenamiento de los modelos, principalmente hemos utilizado dos vectorizaciones, aunque el modelo BERT no necesita vectorización previa, y explicaremos el porque en la sección de implementación de este. La primera, CountVectorizer [18], es un tipo de vectorización que convierte un conjunto de documentos de texto en una representación numérica basada en el recuento de palabras. Cada documento se representa como un vector donde cada elemento corresponde al número de veces que una palabra específica aparece en ese documento.

La segunda vectorización, Tf-Idf, es un tipo de vectorización que combina información sobre la frecuencia de las palabras en un documento con su importancia en el conjunto de documentos. En este caso el termino "documento" sería equivalente a un comentario individual [19].

La ventaja de CountVectorizer es su simplicidad y eficacia en la captura de información básica sobre la frecuencia de palabras en un corpus. Sin embargo, este enfoque no considera la importancia relativa de las palabras en el conjunto de datos y puede generar vectores de alta dimensionalidad [18].

La ventaja de Tf-Idf es su capacidad para destacar términos clave y reducir la influencia de palabras comunes al considerar la frecuencia de término e inversa de documento. Sin embargo, en conjuntos de datos con documentos muy cortos o variabilidad en la longitud (como es nuestro caso), sus beneficios podrían ser limitados, requiriendo ajustes o técnicas adicionales [19].

Otra característica importante sobre la vectorización son los n-gramas. Los n-gramas son secuencias contiguas de n elementos tomados de una muestra de texto o habla. En el contexto del procesamiento del lenguaje natural, como es nuestro caso, estos elementos son palabras. Los n-gramas son utilizados para capturar información sobre la estructura y el significado del lenguaje. Es decir un bigrama cogería las palabras dos a dos y las codificaría en un mismo vector. Esto nos ayuda a añadir contexto a cada palabra y mejorar como el modelo relaciona cada palabra con las demás.

4.2. K-Nearest Neighbours

El algoritmo de los k-vecinos más cercanos es un algoritmo de clasificación basado en aprendizaje supervisado, el cual clasifica nuevos datos basándose en su proximidad a los datos conocidos, asumiendo que datos que son similares tienden a estar más próximos [20]. El funcionamiento del algoritmo es bastante sencillo, primero se computa la distancia entre el nuevo punto (dato) que estamos estudiando con los datos cuya clasificación ya conocemos, nuestro *training set*. Después se eligen los k puntos más cercanos y finalmente a nuestro nuevo punto se le clasifica con el *label* que aparece más veces entre nuestra selección k.

El valor de k influye mucho en el *output* del algoritmo. Tomemos como ejemplo

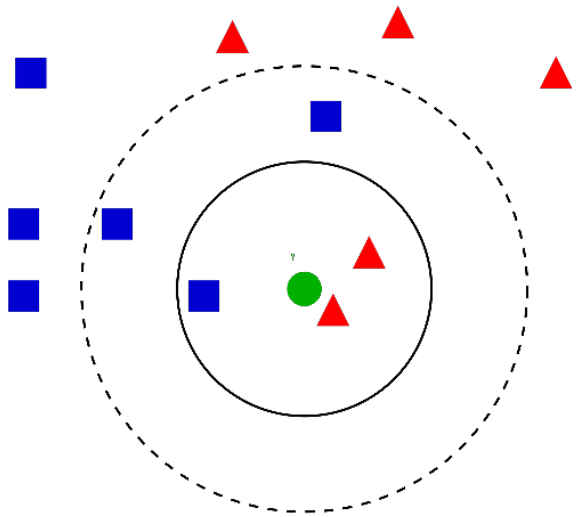

Figura 4.1: K-Nearest Neighbours

la imagen [21] que aparece en la Figura 4.1. En este caso, si se selecciona como k el valor 3 (k = 3), el nuevo dato se clasificará como triángulo rojo ya que de los 3 vecinos 2 son triángulos rojos y uno es cuadrado azul. Sin embargo, si seleccionamos como k el valor 5 (k = 5), entonces el nuevo dato se clasificará como cuadrado azul ya que habrá 3 cuadrados azules frente a 2 triángulos rojos.

Una de las ventajas de este modelo es que no tiene demasiados hiperparámetros, con optimizar k, la métrica para calcular la distancia y el peso de los vecinos es suficiente. No hay ninguna forma de elegir un valor k correcto a priori, la mejor forma es ir probando valores hasta obtener los mejores resultados. Sin embargo podemos tener en cuenta que los valores de k demasiado bajos puede ser sensibles a datos que están alejados del *cluster* de su categoría o incluso puede haber *overfitting*. Por el contrario, si el valor de k es demasiado grande puede hacer que las categorías que no tienen muchos miembros siempre pierdan la fase de votación frente a categorías muy pobladas, es decir, *underfitting* [22]. Los efectos de ir variando el valor de k los podemos apreciar en las Figuras 4.2 y 4.3 [21].

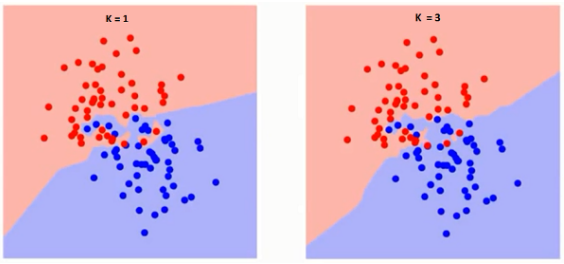

Figura 4.2: k = 1 frente a k = 3

Implementación

A continuación se describe el código empleado para implementar el modelo K nearest Neighbours.

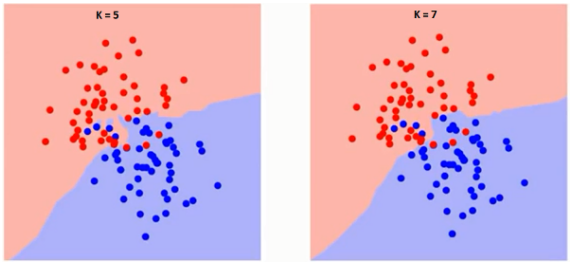

Figura 4.3: k = 5 frente a k = 7

Modulos/librerias utilizadas

En la Tabla 4.1 podemos apreciar los modulos necesarios para implementar el modelo KNN y obtener diferentes métricas de resultados.

Modulo	**Libreria**	**Descripción**
pandas	pandas	Manipulación y análisis de datos de manera eficiente y fácil.
TfidfVectorizer	sklearn.feature_extraction.text	Permite utilizar TfidfVectorizer en Python.
KNeighborsClassifier	sklearn.neighbors	Incluye el clasificador K-Nearest Neighbours.
train_test_split	sklearn.model_selection	Divide *datasets* en subconjuntos de entrenamiento y prueba.
GridSearchCV	sklearn.model_selection	Para implementar la GridSearch.
accuracy_score	sklearn.metrics	Para computar la precisión.
classification_report	sklearn.metrics	Construye un informe que muestra las principales métricas de clasificación.
matplotlib.pyplot	matplotlib	Destinado para gráficos interactivos.
seaborn	seaborn	Funciones de visualización útiles para entender el rendimiento de los modelos.

Tabla 4.1: Módulos utilizados en la implementación del K-Nearest Neighbours

```
df = pd.read_csv('comments.csv', sep='|')
data = df[['TextData', 'Sentiment']]
train_data, test_data, train_labels, test_labels = train_test_split(
    data['TextData'], data['Sentiment'], test_size=0.2, random_state=42
)
```

Figura 4.4: Carga de datos y train-test split

En la Figura 4.4 se encuentra el código para realizar la carga del *dataset* desde el fichero *comments.csv*. Tras la carga se seleccionan las columnas más relevantes para el análisis que en este caso son las columnas de 'TextData'y 'Sentiment'y finalmente se realiza la separación entre datos de entrenamiento y datos de testing. Tras la carga

```
tfidf_vectorizer = TfidfVectorizer(max_features=1000)
train_features = tfidf_vectorizer.fit_transform(train_data)
test_features = tfidf_vectorizer.transform(test_data)
```

Figura 4.5: Vectorización K-Nearest Neighbours

del dataset y la separación de los datos, se procede con la vectorización. Para este modelo se ha empleado una vectorización TF-IDF con 1000 características, como puede observarse en la Figura 4.5. Una vez se ha hecho la vectorización, se procede

```
param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski', 'hamming', 'cosine']
}

knn_classifier = KNeighborsClassifier()

grid_search = GridSearchCV(knn_classifier, param_grid, cv=5)
grid_search.fit(train_features, train_labels)

best_params = grid_search.best_params_
best_knn_classifier = grid_search.best_estimator_

predictions = best_knn_classifier.predict(test_features)
```

Figura 4.6: Grid search y creación del modelo

con el entrenamiento del modelo. Para este modelo se ha implementado una *Grid-search*, en la cual se prueban con diferentes combinaciones de hiperparámetros para dar con la combinación óptima. Tras instanciar el modelo y conseguir la combinación óptima de hiperparámetros se hacen las predicciones a partir del subconjunto de datos destinados al test. Todo este proceso queda ilustrado en la Figura 4.6. Una vez finalizada esta sección sólo queda mostrar los resultados obtenidos, para ello se ha optado por emplear una matriz de confusión, el código para crearla se puede ver en la Figura 4.7, y también una Tabla mostrando las métricas que se computa con el código de la Figura 4.8.

```
conf_matrix = confusion_matrix(test_labels, predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Neutral', 'Positive'], yticklabels=['N
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('k-NN Confusion Matrix')
plt.show()
```

Figura 4.7: Código para crear la matriz de confusión del K-NN

```
accuracy = classification_report(test_labels, predictions, output_dict=True)
accuracy1 = accuracy_score(test_labels, predictions)

accuracy_df = pd.DataFrame(accuracy).transpose()
results_filename = 'knn_hyperparameter_results.csv'
accuracy_df.to_csv(results_filename, index=False)

print(f"Best Hyperparameters: {best_params}")
print(f"Results saved to {results_filename}")
print(f'Accuracy: {accuracy1:.2f}')
print(classification_report(test_labels, predictions))
```

Figura 4.8: Código para mostrar los resultados

4.3. Multinomial Naive Bayes

Este algoritmo probabilístico pertenece a la familia de algoritmos basados en el teorema de Bayes, la etiqueta de *naive* viene dada por la asunción de que las diferentes características son independientes, esto quiere decir que la presencia o ausencia de una característica determinada no tiene relación con la presencia o ausencia de cualquier otra característica. Esta asunción simplifica el cálculo de probabilidades y hace que sea más eficiente en términos de computación [23]. El funcionamiento del algoritmo es relativamente simple. Supongamos que tiene una serie de comentarios divididos entre positivos y negativos. Lo primero sería guardar todas las palabras que aparecen en estos mensajes y asociar la probabilidad de que estas palabras aparezcan en cada tipo de mensaje:

$$p(palabra|clase) = (num\ ocurrencias/total\ palabras) \tag{4.1}$$

Tras esto calculamos las probabilidades a priori de las clases positivo y negativo:

$$p(clase) = (num\ comentarios\ clase/total\ comentarios). \tag{4.2}$$

Seguidamente se calcula la probabilidad de que un comentario pertenezca a una u otra clase. Para lograr esto se multiplican las probabilidades de aparición de cada una de las palabras por la probabilidad a priori de las clases. Un comentario pertenece a la clase para la que obtiene la mayor probabilidad.

Existe una problemática y es el caso en el que en el conjunto de entrenamiento haya palabras que no aparecen en ambas clases. Esto puede causar que comentarios que a priori pertenecerían a una clase acaben siendo clasificados en la contraria. La causa de esto es que si una palabra no aparece en los comentarios del entrenamiento de una clase la probabilidad de que esa palabra aparezca en esa clase sería 0, por tanto da igual las demás que aparezcan en el comentario a clasificar ya que haría que la probabilidad total sea 0 y se clasificaría como de la clase contraria. Para evitar esto se añade al conteo de cada palabra un valor determinado alfa para asegurarnos que nunca haya un 0.

Implementacion

En este apartado procedemos a la explicación del código para la vectorización, entrenamiento del modelo, generación de resultados y representación de los mismos.

Modulos/librerias utilizadas

El la Tabla 4.2 se pueden apreciar todas las librerías y módulos utilizados para el entrenamiento de este modelo, acompañados de una pequeña descripción.

Módulo	Librería	Descripción
pandas	pandas	Manipulación y análisis de datos de manera eficiente y fácil.
CountVectorizer	sklearn.feature_extraction.text	Permite utilizar Count Vectorizer en Python.
MultinomialNB	sklearn.naive_bayes	Incluye el clasificador Naive Bayes para modelos multinomiales.
train_test_split	sklearn.model_selection	Divide *datasets* en subconjuntos de entrenamiento y prueba.
accuracy_score	sklearn.metrics	Para computar la precisión.
classification_report	sklearn.metrics	Construye un informe que muestra las principales métricas de clasificación.
matplotlib.pyplot	matplotlib	Destinado para gráficos interactivos.
seaborn	seaborn	Funciones de visualización útiles para entender el rendimiento de los modelos.

Tabla 4.2: Módulos utilizados en la implementación de Multinomial Naive Bayes

Vectorización utilizada

Para este apartado se ha realizado la vectorización empleando CountVectorizer, con todas las características posibles.

Código

A continuación se muestran los pasos para implementar el modelo. Primero de todo es necesario cargar los datos con los que trabajar desde el CSV con los datos que se han extraido. Para poder entrenar nuestro modelo y comprobar su rendimiento,

es necesario que se realice una separación de los datos en dos subconjuntos, uno de entrenamiento y otro para testear los resultados [24]. El proceso descrito hasta ahora se puede apreciar en la Figura 4.9.

```
df = pd.read_csv('comments.csv', sep='|')
data = df[['TextData', 'Sentiment']]
train_data, test_data, train_labels, test_labels = train_test_split(
    data['TextData'], data['Sentiment'], test_size=0.2, random_state=42
)
```

Figura 4.9: Código para cargar los datos

Una vez se han separado los datos, se procede con la vectorización que podemos apreciar en la Figura 4.10.

```
vectorizer = CountVectorizer()
train_features = vectorizer.fit_transform(train_data)
test_features = vectorizer.transform(test_data)
```

Figura 4.10: Código de la vectorización

Tras la vectorización, llega el momento de crear el modelo, entrenarlo haciendo uso del subconjunto de datos de entrenamiento y finalmente se hacen predicciones sobre el subconjunto de test. Toda esta sección se puede apreciar en la Figura 4.11.

```
naive_bayes_classifier = MultinomialNB()
naive_bayes_classifier.fit(train_features, train_labels)
predictions = naive_bayes_classifier.predict(test_features)
```

Figura 4.11: Código de la creación, entrenamiento y predicciones

Una vez se han hecho las predicciones, es momento de visualizar los resultados. Para ello, se ha optado por emplear una matriz de confusión y una Tabla con diferentes métricas. El código empleado para computar estas representaciones se puede apreciar en la Figura 4.12 y la Figura 4.13.

```
conf_matrix = confusion_matrix(test_labels, predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Neutral', 'Positive'], yticklabels=['N
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Figura 4.12: Código matriz de confusión

```
accuracy = accuracy_score(test_labels, predictions)
classification_report_result = classification_report(test_labels, predictions)
print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:\n', classification_report_result)
```

Figura 4.13: Código Tabla de métricas

4.4. Random Forest

Random Forest es un algoritmo de aprendizaje automático basado en el uso de arboles de decisión, con diferentes conjuntos de entrenamiento, extraídos normalmente del mismo *dataset*, solo que variando la composición de los conjuntos. Una vez construido cada árbol individual, se juntan todos los resultados de cada árbol y mediante un proceso de votación, se calcula el resultado más apto para dicho modelo [25]. En la Figura 4.14 se puede apreciar un esquema simplificado del funcionamiento del modelo [26]. La construcción de este algoritmo sigue los pasos que se describen a continuación.

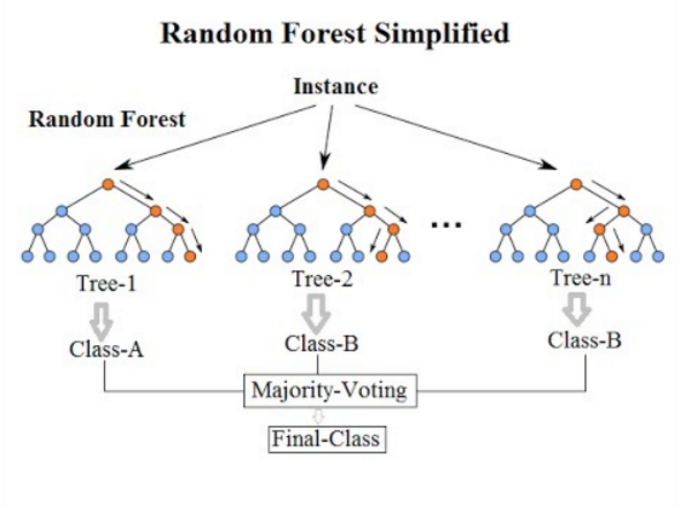

Figura 4.14: Random Forest

4.4.1. Hiperparámetros del modelo

En este modelo hemos utilizado 200 estimadores, los cuales representan 200 árboles de decisión individuales. Es importante destacar que hemos configurado el número mínimo de muestras requeridas para dividir un nodo interno del árbol en 2. Además, hemos establecido que el número mínimo de muestras en una hoja del árbol debe ser 1. La obtención de estos parámetros se realizó utilizando GridSearch.

4.4.2. Pasos en Random Forest

En esta sección se detallan en profundidad los distintos pasos que se realizan durante el entrenamiento de este modelo, desde la selección de los datos de entrenamiento hasta la votación de los distintos árboles generados.

Muestreo con reemplazo

Se toman múltiples muestras aleatorias con reemplazo del conjunto de datos de entrenamiento. Esto implica seleccionar aleatoriamente instancias del conjunto de

datos original, permitiendo que una misma instancia aparezca varias veces o que algunas no aparezcan en absoluto en una muestra específica.

Arboles de decisión

Para cada una de estas muestras, se construye un árbol de decisión. Cada árbol se desarrolla considerando solo un subconjunto aleatorio de las características disponibles en cada división. Esto ayuda a diversificar los árboles y evitar que todos dependan demasiado de las mismas características.

Votación

Una vez que todos los árboles se han construido, para problemas de clasificación se realiza una votación entre ellos para determinar la clase final, mientras que en problemas de regresión se promedian sus predicciones. Para problemas de clasificación tenemos dos opciones:

- Votación Ponderada: Cada árbol emite una predicción de la clase para una instancia dada. La predicción final se determina mediante una votación ponderada, donde se asigna un peso a cada árbol según su precisión en el conjunto de entrenamiento. Los árboles más precisos tienen un mayor peso en la decisión final.
- Clase Mayoritaria: Otra opción común es simplemente tomar la clase que es predicha por la mayoría de los árboles. En este enfoque, cada árbol tiene el mismo peso en la votación, independientemente de su precisión individual.

4.4.3. Validación cruzada

La validación cruzada es una técnica esencial para evaluar el rendimiento de un modelo de aprendizaje automático de manera más robusta y confiable. En nuestro caso, la validación cruzada nos ayuda a estimar cómo se generaliza el rendimiento del modelo a nuevos datos [27]. Para realizar la validación cruzada hemos realizado 10 *folds* o pliegues, es decir, 10 subconjuntos dentro de nuestro conjunto general de datos. Para realizar la validación cruzada realizamos 10 iteraciones (una por pliegue), en la que se usa un pliegue como conjunto de test y el resto de pliegues como conjunto de entrenamiento [28]. Por cada iteración se va cambiando el pliegue de test y se calcula la métrica que queremos evaluar, en el caso de este modelo la puntuación F1. Una vez se calculan todas las puntaciones F1 se calcula la media, y esa será nuestra métrica "generalizada", lo cual nos indica la robustez de nuestro modelo y como funcionará con la introducción de datos nuevos.

4.4.4. Implementación

En esta sección explicaremos la implementación del modelo Random Forest con Python y la extracción de los resultados, así como la vectorización y los módulos utilizados. Cabe destacar que la versión CountVectorizer y la versión TF-IDF son

iguales, lo único que cambia es que en el pipeline [29] se incluye una vectorización u otra.

Modulos/librerias utilizadas

El la Tabla 4.3 se muestran todas las librerías y módulos utilizados para el entrenamiento de este modelo, con una pequeña descripción.

Vectorización utilizada

En este modelo, hemos empleado un CountVectorizer y TfidfVectorizer con un límite máximo de 200 características y un rango de n-gramas que abarca desde 1 hasta 2 (monogramas y bigramas pueden ser utilizados).

En el apartado de resultados compararemos que vectorización funciona mejor para nuestro conjunto de datos y la elegiremos a la hora de comparar todos nuestros modelos entre sí.

Código

Una vez explicados los pasos y herramientas que conforman nuestro modelo Random Forest, podemos explicar un poco por encima nuestro código en Python entrenando el modelos. En la Figura 4.15 podemos observar la primera parte del código, en la que se entrena el modelo en si, en el que se van realizando los siguientes pasos (en orden):

- En primer lugar extraemos el conjunto de datos desde nuestro archivo .CSV y lo almacenamos en un dataframe.
- Creamos un pipeline en el que incluimos el vectorizador (CountVecotirzer o TF-IDF según el caso) y nuestro modelo con los hiperparámetros. El pipeline simplemente encadena la vectorización y el entrenamiento del modelo en un solo paso.
- Dividimos nuestro conjunto de datos en datos de entrenamiento y datos de validación usando *train_ test_ split* (80 % entrenamiento - 20 % test) [30].
- Utilizamos nuestro pipeline para entrenar nuestro modelo con validación cruzada (10 folds) [28].
- Observamos en la Figura 4.16 que obtenemos las puntuaciones de la validación cruzada y la puntación media de esta. Cabe recordar que para la validación cruzada estamos usando la puntuación F1 como métrica objetivo.
- Una vez entrenado el modelo, probamos con nuestro conjunto de datos el rendimiento de nuestro Random Forest. Utilizamos un Classification Report y una matriz de confusión para poder analizar los resultados [31]. Podemos observar el resultado de los reportes de clasificación para ambos modelos en las Figuras 4.18 y 4.17.

Modulo	**Libreria**	**Descripción**
pandas	pandas	Manipulación y análisis de datos de manera eficiente y fácil.
sys	sys	Funciones y variables que interactúan con el sistema operativo.
CountVectorizer	sklearn.feature_extraction.text	Permite utilizar Count Vectorizer en Python.
TfidfVectorizer	sklearn.feature_extraction.text	Permite utilizar Tf-Idf Vectorizer en Python.
RandomForestClassifier	sklearn.ensemble	Incluye el estimador Random Forest para Python.
cross_val_score	sklearn.model_selection	Evalúa una métrica mediante validación cruzada.
train_test_split	sklearn.model_selection	Divide *datasets* en subconjuntos de entrenamiento y prueba.
Pipeline	sklearn.pipelines	Conjunto de transformaciones con un estimador final.
classification_report	sklearn.metrics	Construye un informe que muestra las principales métricas de clasificación.
matplotlib.pyplot	matplotlib	Destinado para gráficos interactivos.
scikitplot	scikitplot	Funciones de visualización útiles para entender el rendimiento de los modelos.

Tabla 4.3: Módulos utilizados en la implementación de Random Forest

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sys
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, train_test_split, cross_val_predict
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
import scikitplot as skplt


filename = sys.argv[1]
comments_df = pd.read_csv(filename, sep='|')

X = comments_df['TextData']
y = comments_df['Sentiment']

pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(max_features=2000, ngram_range=(1, 2))),
    ('classifier', RandomForestClassifier(
        n_estimators=200, min_samples_split=2, min_samples_leaf=1, random_state=42
    ))
])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

cv_scores = cross_val_score(pipeline, X_train, y_train, cv=10, scoring='f1_macro', n_jobs=-
```

Figura 4.15: Código del modelo Random Forest (Entrenamiento)

```python
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", cv_scores.mean())

pipeline.fit(X_train, y_train)

test_score = pipeline.score(X_test, y_test)
print("Test Set Accuracy:", test_score)

y_pred_test = pipeline.predict(X_test)

print("Classification Report:")
print(classification_report(y_test, y_pred_test))

print("Confusion Matrix:")
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test, normalize=False)
plt.show()
```

Figura 4.16: Código del modelo Random Forest (Métricas)

```
Cross-Validation Scores: [0.64611109 0.66166935 0.67376875 0.6654968  0.65318086 0.6358481
 0.67622999 0.63370147 0.67638078 0.65432995]
Mean Cross-Validation Score: 0.6576717137915316
Test Set Accuracy: 0.731614135625597
Classification Report:
              precision    recall  f1-score   support

    Negative       0.80      0.35      0.49       547
     Neutral       0.68      0.77      0.72       429
    Positive       0.74      0.91      0.81      1118

    accuracy                           0.73      2094
   macro avg       0.74      0.67      0.67      2094
weighted avg       0.74      0.73      0.71      2094
```

Figura 4.17: Métricas tras ejecución con Tf-Idf

```
Cross-Validation Scores: [0.66559985 0.69614019 0.67515828 0.66615431 0.65479414 0.67837761
 0.68532659 0.66132788 0.71265441 0.66984204]
Mean Cross-Validation Score: 0.6765375302683585
Test Set Accuracy: 0.7402101241642789
Classification Report:
              precision    recall  f1-score   support

    Negative       0.76      0.40      0.53       547
     Neutral       0.68      0.76      0.72       429
    Positive       0.76      0.90      0.82      1118

    accuracy                           0.74      2094
   macro avg       0.73      0.69      0.69      2094
weighted avg       0.74      0.74      0.72      2094
```

Figura 4.18: Métricas tras ejecución del modelo con CountVectorizer

4.5. Regresión Logística Multinomial

La Regresión Logística Multinomial es una extensión de la Regresión Logística binomial cuando se trata de problemas de clasificación con más de dos categorías [32]. Mientras que la Regresión Logística binomial se utiliza para predecir la probabilidad de pertenencia a una de dos categorías, la Regresión Logística Multinomial maneja situaciones donde hay más de dos categorías mutuamente excluyentes.

4.5.1. Regresión Logística Binaria

La regresión logística es un método estadístico utilizado para modelar la probabilidad de que un evento binario ocurra como una función de una o más variables predictoras. Es particularmente útil cuando la variable dependiente es categórica y tiene dos categorías, como *sí o no*, *éxito o fracaso*, *positivo o negativo*, como es nuestro caso. Pero como nosotros añadimos un clase *neutra* para añadir mas profundidad al entrenamiento de los datos, necesitaremos usar una regresión multiclase.

En la regresión logística, el modelo matemático utiliza la función logística (también conocida como función sigmoide, ilustrada en la Figura 4.19.) para transformar una combinación lineal de las variables predictoras en una variable contenida entre 0 y 1, que determina la proximidad de la información a una clase u otra. Como tenemos 3 clases, tendremos que hacer algo diferente, ya que para más de dos clases es mejor utilizar una función softmax, que es una generalización de la función sigmoide, como se explicará más adelante.

$$f(x) = \frac{1}{1 + e^{-x}}$$

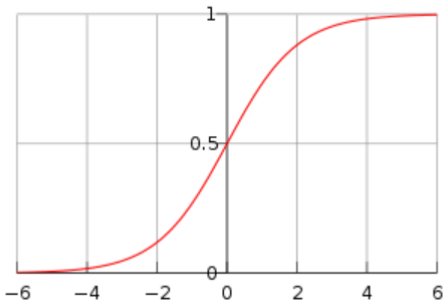

Figura 4.19: Función sigmoide

4.5.2. Pasos de la regresión logística multinomial

A continuación se describen los distintos pasos que se realizan para completar el entrenamiento de nuestros datos con regresión logística.

Definición del Problema

Este método se utiliza cuando se tiene una variable dependiente categórica con tres o más categorías. Por ejemplo, puede ser utilizado para predecir la categoría de un producto entre *bajo, medio o alto* basándose en diversas características. En nuestro caso tenemos 3 clases que definen el sentimiento del comentario: *Positivo, Negativo, y Neutro.*

Función Softmax

La función softmax se define para cada clase i de la siguiente manera [33]:

$$P(y = i|\mathbf{x}) = \frac{e^{\mathbf{w}_i \cdot \mathbf{x}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j \cdot \mathbf{x}}}$$

Donde:

- $P(y = i|\mathbf{x})$ es la probabilidad de que la observación $\mathbf{x}$ pertenezca a la clase i.
- K es el número total de clases.
- $\mathbf{w}_i$ es el vector de pesos asociados a la clase i.

En este contexto, $\mathbf{w}_i \cdot \mathbf{x}$ representa la combinación lineal de las características de la observación $\mathbf{x}$ ponderadas por los pesos asociados a la clase i. La función softmax normaliza estas combinaciones lineales elevando y dividiendo por la suma de las exponenciales para obtener probabilidades que suman 1.

La predicción final para una observación $\mathbf{x}$ será la clase con la probabilidad más alta según la función softmax.

En términos más sencillos, la función softmax asigna probabilidades a cada clase y permite la clasificación de instancias en una de las clases.

Estimación de Parámetros

En regresión logística, la estimación de parámetros implica encontrar los coeficientes que maximizan la función de verosimilitud, la cual mide la probabilidad de observar los datos dados los parámetros. Se utiliza un algoritmo de optimización, como el descenso de gradiente, para ajustar los coeficientes iterativamente.

En este caso los parámetros del modelo que nos interesan son el numero de iteraciones y el *solver*. En nuestro caso realizamos 500 iteraciones y usamos el *solver lbfgs (Limited-memory Broyden-Fletcher-Goldfarb-Shanno)*.

El parámetro *max iter* (número máximo de iteraciones) se refiere al límite máximo de iteraciones permitido para que el algoritmo de optimización converja y encuentre los parámetros óptimos del modelo.

El *solver* se refiere al algoritmo utilizado para optimizar los parámetros del modelo durante el proceso de entrenamiento. Diferentes *solvers* implementan diferentes algoritmos de optimización.

En el contexto de regresión logística y otros modelos lineales generalizados, el *solver lbfgs* se utiliza para encontrar los coeficientes del modelo que maximizan la función de verosimilitud. En otras palabras, se utiliza para encontrar los parámetros que hacen que los datos observados sean más probables bajo el modelo logístico.

4.5.3. Implementación

A continuación explicamos la implementación realizada con Regresión Logística, así como la vectorización utilizada y sus parámetros más relevantes.

Módulos/librerías utilizadas

El la Tabla 4.4 se muestran todas las librerías y módulos utilizados para el entrenamiento de este modelo, con una pequeña descripción.

Módulo	Librería	Descripción
pandas	pandas	Manipulación y análisis de datos de manera eficiente y fácil.
sys	sys	Funciones y variables que interactúan con el sistema operativo.
CountVectorizer	sklearn.feature_extraction.text	Permite utilizar Count Vectorizer en Python.
LogisticRegression	sklearn.linear_model	Incluye el modelo de regresión logística para Python.
cross_val_score	sklearn.model_selection	Evalúa una métrica mediante validación cruzada.
train_test_split	sklearn.model_selection	Divide *datasets* en subconjuntos de entrenamiento y prueba.
Pipeline	skelearn.pipelines	Conjunto de transformaciones con un estimador final.
classification_report	sklearn.metrics	Construye un informe que muestra las principales métricas de clasificación.
accuracy_score	sklearn.metrics	Puntuación de exactitud.
matplotlib.pyplot	matplotlib	Destinado para gráficos interactivos.
scikitplot	scikitplot	Funciones de visualización que son útiles para entender el rendimiento de los modelos.

Tabla 4.4: Módulos utilizados en la implementación de regresión logística

Vectorización utilizada

En este modelo también hemos usado CountVectorizer, con 2000 como el numero máximo de atributos y un rango de n-gramas entre 1 y 2.

Código

La implementación de este modelo es prácticamente igual que la implementación del Random Forest, ya que al usar un *pipeline* es mucho más fácil abstraer el funcionamiento general del entrenamiento, cambiando solo el modelo a entrenar en nuestro pipeline. Esto lo podemos observar en la Figura 4.21, viendo que la implementación es, en efecto, igual que con Random Forest. Aunque si bien es cierto que el entrenamiento es muy similar, con la regresión logística podemos hacer más cosas a la hora de hacer un análisis más exhaustivo de los resultados. Esa parte del código no la incluimos en esta sección ya que profundizaremos mas sobre este en la sección de análisis de resultados. A continuación un resumen de los pasos realizados:

- Primero extraemos el archivo .csv a través de los argumentos del script y guardamos el conjunto de datos en un dataframe.
- Creamos un pipeline con nuestro vectorizador (CountVectorizer) y el modelo con los parámetros. Estos parámetros fueron optimizados usando GridSearch, pero en nuestro código final no aparece por el alto consumo de tiempo en calcularlos.
- Dividimos nuestro conjunto de datos en datos de entrenamiento y datos de validación usando *train_test_split* (80 % entrenamiento - 20 % test).
- Utilizamos nuestro pipeline para entrenar nuestro modelo con validación cruzada (10 pliegues) [28].
- Una vez entrenado el modelo, obtendremos las métricas que reflejan el rendimiento del modelo con un conjunto de validación. Esta información se refleja en el Classification Report (Figura 4.22) y en la matriz de confusión [31].

```
Cross-Validation Scores: [0.70355326 0.70878447 0.7115865  0.72184086 0.71456588 0.70170069
 0.7038219  0.70422738 0.72150926 0.68604806]
Mean Cross-Validation Score: 0.7077638255234701
Classification Report:
              precision    recall  f1-score   support

    Negative       0.69      0.58      0.63       547
     Neutral       0.64      0.75      0.69       429
    Positive       0.82      0.83      0.82      1118

    accuracy                           0.75      2094
   macro avg       0.71      0.72      0.71      2094
weighted avg       0.75      0.75      0.75      2094
```

Figura 4.20: Métricas de la regresión logística

```python
import pandas as pd
import sys
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import scikitplot as skplt

filename = sys.argv[1]
comments_df = pd.read_csv(filename, sep='|')

X = comments_df['TextData']
y = comments_df['Sentiment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

pipeline = Pipeline([
    ('vectorizer', CountVectorizer(max_features=2000, ngram_range=(1, 2))),
    ('classifier', LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=500, random_state=42))
])

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)
```

Figura 4.21: Código del modelo Regresión Logística Multiclase (Entrenamiento)

```python
print("Test Set Accuracy:", test_score)

cv_scores = cross_val_score(pipeline, X_train, y_train, cv=10, scoring='f1_macro', n_jobs=-1)
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", cv_scores.mean())

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Figura 4.22: Código del modelo Regresión Logística Multiclase (Métricas)

4.6. Support Vector Machine

Las *Support Vector Machine* (SVM) son un conjunto de métodos de aprendizaje supervisado utilizados para la clasificación, regresión y detección de *outliers*. Son especialmente potentes para espacios de características de alta dimensión y cuando la relación entre las clases no es lineal. Las SVM funcionan encontrando el hiperplano que mejor divide un conjunto de datos en clases, es decir, el hiperplano con mayor margen entre las clases en el espacio de características. Además, a la hora de definir el hiperplano, sólo se consideran los ejemplos de entrenamiento de cada clase que caen justo en la frontera de dichos márgenes. Estos ejemplos reciben el nombre de vectores de soporte [34].

Las SVM son fundamentalmente clasificadores binarios, es decir, están diseñadas para distinguir entre dos clases. Sin embargo, existen estrategias para extender el uso de SVM a problemas multiclase como el nuestro.

4.6.1. SVC

Como en nuestro caso nos centramos en la clasificación de las muestras, tomaremos la versión de SVM que realiza esta tarea específicamente. Se trata de SVC(C-Support Vector Classification). En multiclase, SVC aplica internamente una estrategia *uno contra uno* (OvO), entrenando un clasificador para cada par de clases, tomando como resultado los valores del que mayor confianza ofrezca.

4.6.2. Uno contra el Resto (OvR)

Utilizando este enfoque, entrenaremos un clasificador por cada clase, donde cada una se encarga de distinguir una clase del resto. Durante la clasificación, el modelo que ofrezca la mayor confianza de los tres en su predicción será el que determinará la clase final de la entrada que será el modelo SVC entrenado previamente. De esta manera combinamos el uso tanto del enfoque OvO como OvR.

4.6.3. Tipos de Kernel

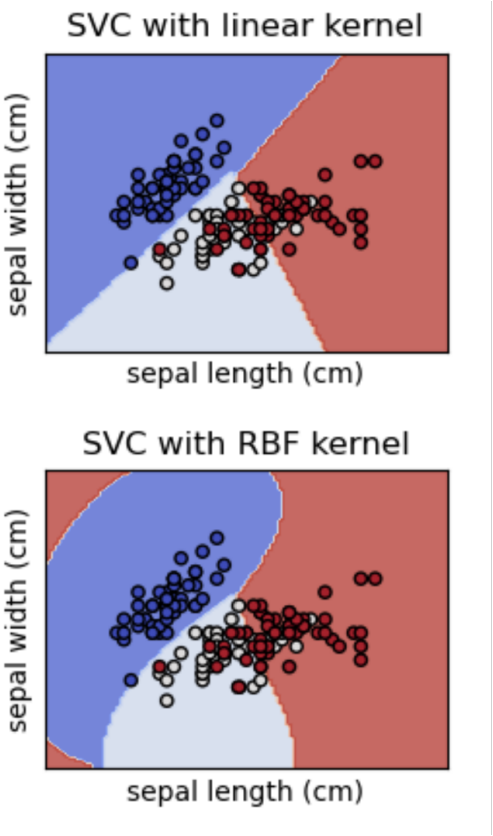

Figura 4.23: Tipos de Kernel para SVM [1]

Distinguimos dos tipos de kernel:

- Kernel Lineal
  - Ideal para datos que son linealmente separables.
  - El hiperplano de decisión es una línea que separa las clases con el margen más amplio.
  - Menos complejidad computacional.
- Kernel RBF
  - Puede manejar datos que no son linealmente separables.
  - Encuentra un hiperplano en una dimensión superior que proyecta los datos de manera que sean separables.
  - Más flexible para capturar relaciones no lineales.

Estos *kernels* están representados de forma gráfica en la Figura 4.23.

4.6.4. Ventajas de las SVM

Las SVM son efectivas en espacios con muchas dimensiones, como cuando tienes muchos atributos, asi como en casos donde el número de dimensiones es mayor que el número de muestras. La función del núcleo puede ser cambiada para adaptarse mejor al problema. Utilizan un subconjunto de puntos de entrenamiento (vectores de soporte), por lo que son también eficientes en términos de memoria.

4.6.5. Implementación

A continuación mostraremos el código con el que implementamos los modelos SVM con distinto enfoque.

Módulos/librerías utilizadas

El la Tabla 4.5 se muestran todas las librerías y módulos utilizados para el entrenamiento de este modelo, con una pequeña descripción.

Vectorización utilizada

Para la vectorización de los datos utilizamos el código que aparece en la Figura 4.24.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils.class_weight import compute_class_weight
from sklearn.feature_extraction.text import TfidfVectorizer

# Carga de datos y preprocesamiento
file_path = 'comments.csv'
data = pd.read_csv(file_path, sep='|')


# Vectorización de los comentarios usando TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X = tfidf_vectorizer.fit_transform(data['Comment'])

y = data['Sentiment'].values

# División del dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Codificación de las etiquetas para poder usar compute_class_weight
label_encoder = LabelEncoder()
encoded_y_train = label_encoder.fit_transform(y_train)

# Calcula los pesos de las clases para abordar el desequilibrio
class_weights = compute_class_weight('balanced', classes=np.unique(encoded_y_train), y=encoded_y_train)

# Mapea los pesos a las etiquetas codificadas
weights = {i: class_weights[i] for i in range(len(np.unique(encoded_y_train)))}

X_train.shape, X_test.shape, y_train.shape, y_test.shape, weights
```

Figura 4.24: Código para la vectorización y preparación del *dataset* de los modelos SVM

En este fragmento de código cargamos nuestro archivo *comments.txt* y obtenemos una columna con los comentarios vectorizados utilizando *TfidVectorizer*. Seguidamente preparamos los *dataset* de entrenamiento con el comentario vectorizado y la etiqueta previamente predicha de manera aleatoria con una proporción de $70-30\,\%$. Además calculamos el peso de las clases de sentimientos y finalmente preparamos los *dataset* de entrenamiento y validación que utilizaremos en ambos modelos SVM. Podemos ver el código que implementa este proceso en la Figura 4.24. A continuación veremos el código utilizado para la implementación de dos modelos SVM en la tarea de análisis de sentimiento. Uno de ellos implementado con el enfoque *OvO* y otro implementando *OvR* sobre el modelo creado con *OvO*.

Módulo	Librería	Descripción
pandas	pandas	Manipulación y análisis de datos de manera eficiente y fácil.
numpy	numpy	Computación científica en Python.
train_test_split	sklearn.model_selection	Divide *datasets* en subconjuntos de entrenamiento y prueba.
GridSearchCV	sklearn.model_selection	Permite la búsqueda de parámetros para optimizar modelos.
SVC	sklearn.svm	Implementa (SVM) para clasificación.
OneVsRestClassifier	sklearn.multiclass	Estrategia para realizar clasificación multiclase.
StandardScaler	sklearn.preprocessing	Estandariza características.
LabelEncoder	sklearn.preprocessing	Codifica etiquetas con valor entre 0 y n_clases-1.
compute_class_weight	sklearn.utils.class_weight	Pesos de las clases.
classification_report	sklearn.metrics	Informe de las principales métricas de clasificación.
confusion_matrix	sklearn.metrics	Calcula la matriz de confusión.
matplotlib.pyplot	matplotlib	Destinado para gráficos interactivos y estáticos.
TfidfVectorizer	sklearn.feature	Convierte una colección de datos en una matriz TF-IDF.

Tabla 4.5: Módulos utilizados en la implementación del modelo SVM

SVC

Comenzamos realizando la búsqueda de los valores ideales para los hiperparámetros de nuestro modelo SVC [35; 36] utilizando *GridSearch*. El proceso puede apreciarse en la Figura 4.25. Creamos el modelo con los parámetros obtenidos de la búsqueda con *GridSearch* y lo entrenamos con los *dataset* de entrenamiento.

```python
# Codificación de las etiquetas
label_encoder = LabelEncoder()
encoded_y_sample = label_encoder.fit_transform(y_train)

# Configuración de GridSearchCV
param_grid = {
    'C': [0.1,1,10, 100, 500],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto'],
    'class_weight': [weights]  # Añadir pesos de clase aquí
}

# Creación del modelo y búsqueda de hiperparámetros
model = SVC()
grid_search = GridSearchCV(model, param_grid, cv=2, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, encoded_y_sample)

# Mejores hiperparámetros encontrados
print("Mejores hiperparámetros:", grid_search.best_params_)
# Extrae los mejores parámetros y elimina el prefijo 'estimator__'
svc_params = {k: v for k, v in grid_search.best_params_.items()}
```

Figura 4.25: Código para la búsqueda de parámetros con GridSearch en el modelo SVM con SVC

```python
# Crea y entrena el modelo SVM con los parámetros encontrados
model = SVC(**svc_params)
model.fit(X_train, encoded_y_train)

# Evalúa el modelo
encoded_y_test = label_encoder.transform(y_test)
y_pred = model.predict(X_test)
y_pred = label_encoder.inverse_transform(y_pred)  # Convierte las predicciones codificadas de vuelta a etiquetas originales
```

Figura 4.26: Código del modelo SVM con SVC

Después, lo evaluamos con el *dataset* de validación como muestra la Figura 4.26 para después mostrar los resultados de la manera que refleja el código representado en la Figura 4.27.

```python
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
# Calcula la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Etiquetas personalizadas para las clases
class_labels = ['Negativo', 'Neutro', 'Positivo']

# Grafica la matriz de confusión como un mapa de calor
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.title('Matriz de Confusión')
plt.xlabel('Predicho')
plt.ylabel('Verdadero')
plt.show()
```

Figura 4.27: Código para la visualización de los resultados del modelo SVM con SVC

OvR

Para el modelo *SVM* con enfoque *OneVsRest* repetiremos el proceso anterior pero implementando la función de *OneVsRestClasiffier* [37] al modelo *SVC*.

Comenzamos buscando los valores ideales para los hiperparámetros de nuestro modelo con *GridSearch* como puede apreciarse en la Figura 4.28.

De igual manera que en el modelo anterior, creamos el modelo con los parámetros obtenidos de la búsqueda con *GridSearch* y lo entrenamos con los *dataset* de entrenamiento. Después, lo evaluamos con el *dataset* de validación como muestra la Figura 4.29 y obtendremos la representación de los resultados con el código representado en la Figura 4.30.

```python
# Codificación de las etiquetas
label_encoder = LabelEncoder()
encoded_y_sample = label_encoder.fit_transform(y_train)
# Configuración de GridSearchCV
param_grid = {
    'estimator__C': [0.1,1,10, 100, 500],
    'estimator__kernel': ['linear', 'rbf'],
    'estimator__gamma': ['scale', 'auto'],
    'estimator__class_weight': [weights]  # Añadir pesos de clase aquí
}

# Creación del modelo y búsqueda de hiperparámetros
model = OneVsRestClassifier(SVC())
grid_search = GridSearchCV(model, param_grid, cv=3, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, encoded_y_sample)

# Mejores hiperparámetros encontrados
print("Mejores hiperparámetros:", grid_search.best_params_)
# Extrae los mejores parámetros y elimina el prefijo 'estimator__'
ovr_params = {k.replace('estimator__', ''): v for k, v in grid_search.best_params_.items()}
```

Figura 4.28: Código de búsqueda de parámetros con GridSearch para el modelo SVM con OvR

```python
# Crea y entrena el modelo SVM con los parámetros encontrados
model = OneVsRestClassifier(SVC(**ovr_params), n_jobs=-1)
model.fit(X_train, encoded_y_train)

# Evalúa el modelo
encoded_y_test = label_encoder.transform(y_test)
y_pred = model.predict(X_test)

# Convierte las predicciones codificadas de vuelta a etiquetas originales
y_pred = label_encoder.inverse_transform(y_pred)
```

Figura 4.29: Código del modelo SVM con OvR

```python
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
# Calcula la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Etiquetas personalizadas para las clases
class_labels = ['Negativo', 'Neutro', 'Positivo']

# Grafica la matriz de confusión como un mapa de calor
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.title('Matriz de Confusión')
plt.xlabel('Predicho')
plt.ylabel('Verdadero')
plt.show()
```

Figura 4.30: Código para la visualización de los resultados

4.7. BERT

El modelo NLP BERT (Bidirectional Encoder Representation for Transformers) es un modelo desarrollado por Google y considerado uno de los mejores en tareas de comprensión del lenguaje, análisis de sentimientos y traducción automática.

Como podemos ver en la Figura 4.31, la arquitectura del modelo BERT tiene dos variantes: BERTBase (con 12 capas en el stack del Encoder) y BERTLARGE (con 24 capas). Estos modelos son excepcionalmente potentes debido a su profundidad y complejidad (con 110M y 340M de parámetros respectivamente). Para nuestro estudio utilizaremos BERTBase ya que es suficientemente potente para nuestra tarea y su complejidad computacional es considerablemente menor. A continuación profundizaremos en los conceptos clave para entender el funcionamiento y la potencia de los modelos BERT [38].

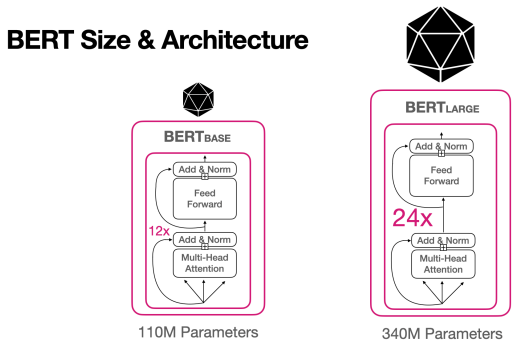

Figura 4.31: Arquitectura y tamaño de los modelos BERT [2]

4.7.1. Arquitectura Transformer

BERT se fundamenta en la arquitectura Transformer. Los Transformers son modelos que se basan en mecanismos de atención para capturar contextos globales de las palabras en una oración. La ventaja más destacable de esta arquitectura para nuestro caso es su mecanismo de atención, el cual permite al modelo ponderar por partes la secuencia de entrada lo cual resulta en un mejor manejo de las dependencias de largo alcance. Este enfoque permite al modelo considerar cada palabra en relación con todas las otras palabras de la oración, proporcionando un entendimiento más profundo del contexto y la semántica del texto. La arquitectura original de Transformer utiliza un codificador para procesar la entrada al modelo y un decodificador que genera la salida. Sin embargo, BERT prescinde del decoder y solo hace uso de encoders ya que el fin principal de BERT es la comprensión del lenguaje, no la generación del mismo.

4.7.2. Entrenamiento Bidireccional

A diferencia de los modelos anteriores de NLP que procesaban el texto de manera unidireccional, BERT analiza los textos de manera bidireccional, lo que significa que el modelo considera tanto el contexto anterior de una palabra como el posterior. Este enfoque bidireccional permite a BERT hacer un entendimiento más matizado y completo del texto.

4.7.3. Preentrenamiento y Afinamiento (Fine-Tuning)

El proceso de entrenamiento de BERT se divide en dos fases. Inicialmente, el modelo se preentrena en un vasto corpus de texto con dos tareas principales: el Modelado de Lenguaje Enmascarado (MLM) y la Predicción de la Siguiente Oración (NSP). En la tarea MLM, al modelo se le presentan oraciones con algunas palabras ocultas, y debe predecir estas palabras basándose en el contexto. La tarea NSP implica predecir si una oración es la continuación lógica de otra. Una vez preentrenado, BERT puede ser afinado con datos específicos para tareas concretas, como el análisis de sentimientos. Esta fase de afinamiento ajusta los parámetros de BERT a las peculiaridades del conjunto de datos objetivo, lo que mejora significativamente su

rendimiento en tareas específicas.

4.7.4. Tokens [CLS] y [SEP] en BERT

Los modelos BERT utilizan *tokens* especiales, [CLS] y [SEP], que resultan cruciales en la comprensión y estructuración de los datos de entrada.

El *token* de clasificación ([CLS]), se coloca al inicio de cada entrada y sirve como un agregado representativo de toda la secuencia para tareas de clasificación como la nuestra.

El *token* de separación ([SEP]) se utiliza para diferenciar segmentos distintos dentro de la misma entrada. Esto es particularmente importante para tareas que involucran múltiples entradas, como la tarea de Predicción de la Siguiente Oración (NSP) durante el preentrenamiento de BERT.

La Figura 4.32 ilustra cómo se incorporan los *tokens* [CLS] y [SEP] dentro de la estructura de entrada de BERT. Cada token es sumado con su correspondiente incrustación posicional y de segmento,las cuales sirven para entender la posición del token en la oración y la pertenencia a un segmento de la entrada respectivamente, antes de ser procesado por las capas del codificador de BERT. La salida del último codificador correspondiente al *token* [CLS] será la utilizada para las tareas de clasificación puesto que representa la secuencia de entrada completa y contextualizada.

4.7.5. Softmax

Se trata de una función de activación que obtiene una distribución de probabilidades a partir de un vector numérico dado. Esta función es normalmente utilizada como capa de clasificación al final de una red neuronal, especialmente en problemas multiclase.

Como vemos en la Figura 4.32, la función softmax recoge un vector que represente los valores de salida de la red neuronal denominados *logits* y lo convierte en una probabilidad para cada elemento del vector. *Logit* es un vector de tamaño *[Nº de muestras][Nº de clases]* que representa la puntuación dada a cada clase para cada comentario en estudio. En nuestro caso el vector de *logits* será una representación de la salida correspondiente al *token* [CLS]. Softmax toma esas puntuaciones y recrea una distribución normalizada de las puntuaciones para poder interpretar los valores para cada etiqueta como una probabilidad.

4.7.6. AdamW

AdamW es un optimizador basado en Adam (Adaptative Moment Estimation) altamente reconocido dado su manejo eficiente de gradientes en el entrenamiento de una red neuronal. La función principal del optimizador es el uso de decaimiento de peso para evitar la equivocación en el entrenamiento de un modelo debido al sobre-ajuste de clases en las muestras. La diferencia que tiene AdamW sobre su versión original reside en la manera en la que se aplica el decaimiento de peso. AdamW, a diferencia de Adam, separa el proceso de optimización de gradientes

en el entrenamiento de la regularización de los pesos aplicando directamente la regularización a los pesos del modelo sin afectar a la actualización de gradientes.

En general, AdamW, nos proporcionará una forma consistente y efectiva de aplicar la regularización durante el entrenamiento de nuestro modelo.

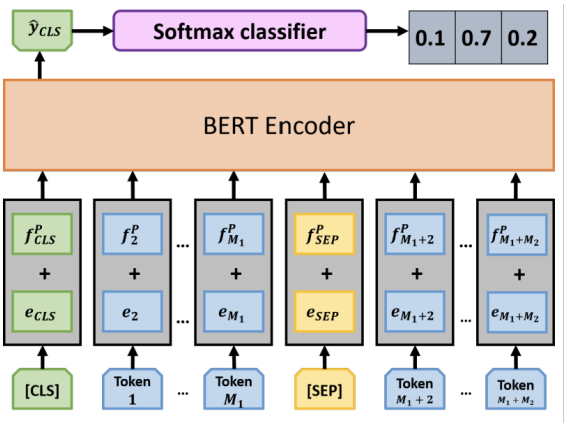

Figura 4.32: Arquitectura de los modelos BERT para tareas de clasificación [3]

4.7.7. Implementación

En esta sección veremos la implementación del modelo BERT en detalle.

Módulos/librerías utilizadas

El la Tabla 4.6 se muestran todas las librerías y módulos utilizados para el entrenamiento de este modelo, con una pequeña descripción.

Vectorización

En el caso de modelos BERT no es necesario utilizar vectorización adicional previa ya que estos modelos son capaces de procesar el significado y contexto de las palabras por su cuenta y de manera más eficaz y precisa que la vectorización tradicional. A diferencia de métodos como *CountVectorizer* o *TF-IDF* en los que se le asigna una representación fija a cada *token* o palabra en modelos BERT utiliza incrustaciones (*embeddings*) dinámicas para reflejar el contexto de la palabra en cada caso de uso distinto.

Código

A continuación procedemos a mostrar el código utilizado para analizar los sentimientos de nuestros *dataset* con un modelo BERT-base [39].

Para mejorar el rendimiento de la ejecución de nuestro código y aprovechar el uso de tensores en modelos BERT utilizaremos CUDA [40]. Comenzamos configurando pytorch para que utilice la GPU del dispositivo en el que se ejecute en caso de estar disponible (Figura 4.33). Además utilizamos la optimización de CUDNN [41]

```
# Configuración de la GPU y optimizaciones de cudnn
torch.backends.cudnn.benchmark = True
# Configuración del dispositivo
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Figura 4.33: ConFigurar uso de GPU

especialmente desarrollada para la aceleración de redes neuronales profundas, cuya guía de instalación se puede encontrar en [42].

```
# Cargar el modelo y el tokenizador
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=3)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model.to(device)
print("Modelo cargado exitosamente.")
```

Figura 4.34: Código para cargar el modelo y tokenizador de BERT-base

La Figura 4.34 refleja la carga del modelo con el número de clases y el tokenizador de BERT-base en las variables *model* y *tokenizer* para después cargar el modelo en el dispositivo de pytorch.

```
# Carga de datos
df = pd.read_csv('comments.csv', sep='|')
comments = df['TextData'].apply(lambda x: ' '.join(eval(x)))
labels = df['Sentiment'].map({'Negative': 0, 'Neutral': 1, 'Positive': 2})
print("Datos cargados exitosamente.")
```

Figura 4.35: Código para cargar los datos de comments.txt

Continuamos cargando nuestro *dataset* y las columnas que contienen los comentarios tokenizados y las etiquetas del sentimiento predicho mapeadas numéricamente en sus respectivas variables como se representa en la Figura 4.35.

```
# Tokenización y preparación de los datos
max_length = 128  # Tamaño de la secuencia
batch_size = 16  # Tamaño del lote
input_ids = tokenizer(comments.tolist(), padding=True, truncation=True, max_length=max_length, return_tensors='pt')['input_ids
attention_masks = tokenizer(comments.tolist(), padding=True, truncation=True, max_length=max_length, return_tensors='pt')['att
labels = torch.tensor(labels.values)
print("Datos preparados exitosamente.")
```

Figura 4.36: Código para preparar los datos de entrada al modelo

Procedemos preparando los datos de entrada del modelo (Figura 4.36). Primero ajustamos el tamaño del lote y el tamaño máximo de la secuencia de entrada.

A continuación, obtenemos los *ids* de los token de nuestros comentarios ajustados a la longitud máxima determinada por la variable *max_ length* en el vocabulario del tokenizador así como la máscara de atención de nuestros tokens como tensores. La máscara de atención es especialmente importante en casos en los que el número de token de un comentario es menor que *max_ length* dado que se añaden tokens de relleno para completar el tensor de entrada puesto que BERT espera un tamaño uniforme en las secuencias de entrada y estos tokens de relleno deben ser tratados como tal. Por último obtenemos en forma de tensor las etiquetas para los sentimientos previamente mapeadas a valores numéricos.

La Figura 4.37 muestra como creamos un *TensorDataset* con los tensores de los ids, las máscaras de atención y las etiquetas obtenidos anteriormente. Después

```
# Crear TensorDataset y dividir en conjuntos de entrenamiento y validación
dataset = TensorDataset(input_ids, attention_masks, labels)
train_size = int(0.8 * len(dataset))  # Cambiado a 80-20 split
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
# Crear DataLoaders para la carga de datos en paralelo
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=8)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=8)
```

Figura 4.37: Código para dividir *dataset* de entrenamiento y evaluación

dividimos aleatoriamente este *dataset* en un conjunto de entrenamiento y otro de validación con una proporción de $80-20\,\%$ y creamos los *DataLoaders* del tensor de entrenamiento y validación para obtener los datos en lotes de manera iterativa en ambas fases. Esto nos permite realizar un entrenamiento mucho más eficiente.

```
# Optimizador y Scheduler
optimizer = AdamW(model.parameters(), lr=3e-5, eps=1e-8)  # Ajustamos la tasa de aprendizaje
epochs = 8  # número de épocas
total_steps = len(train_loader) * epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=total_steps)
print("Optimizador creado exitosamente.")
```

Figura 4.38: Código para crear el optimizador y el scheduler para el entrenamiento y evaluación por épocas

Para el cálculo de pesos y la actualización de los mismos durante el entrenamiento creamos el optimizador *AdamW*. Después, elegimos el número de épocas que queremos para la mejora del modelo y creamos un scheduler con el optimizador para adaptar la tasa de aprendizaje en función de como evolucione el modelo en el entrenamiento. El código para este paso se halla en la Figura 4.38.

```
# Función de evaluación
def evaluate(model, val_loader):
    model.eval()
    total_eval_loss = 0
    predictions, true_labels = [], []

    for batch in val_loader:
        b_input_ids, b_input_mask, b_labels = [t.to(device) for t in batch]
        with torch.no_grad():
            outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)

        total_eval_loss += outputs.loss.item()
        logits = outputs.logits
        probs = softmax(logits.detach().cpu().numpy(), axis=1)
        pred_labels = np.argmax(probs, axis=1)
        label_ids = b_labels.to('cpu').numpy()

        predictions.extend(pred_labels)
        true_labels.extend(label_ids.flatten())

    avg_val_accuracy = accuracy_score(true_labels, predictions)
    avg_val_loss = total_eval_loss / len(val_loader)

    return avg_val_accuracy, avg_val_loss
```

Figura 4.39: Código para dividir *dataset* de entrenamiento y evaluación

Para poder evaluar el modelo con *BERT* creamos la función que aparece en la Figura 4.39, *evaluate(model,val_ loader)* . El funcionamiento principal de esta función consiste en calcular, iterando por lotes, la mayor probabilidad de cada comentario de pertenecer a una clase para un modelo y un *TensorDataset* de validación dados. Normaliza los *logits* resultantes de la evaluación del modelo a través de la función *softmax* y toma el valor máximo para cada caso. A su vez calcula la función de pérdida de cada lote y la añade a la variable *total_ eval_ loss* para después calcular la media de la función de pérdida para el modelo evaluado.

A la hora de entrenar el modelo decidimos implementar el entrenamiento por lotes del *TensorDataset* de entrenamiento *train_ loader*. En cada paso desglosamos el lote

en las entradas que alimentaremos al modelo y reseteamos los *gradientes*. Después calculamos la función de pérdida *loss* y sus gradientes utilizando *backpropagation* con la llamada *loss.backward()* que seran utilizados por el *optimizador*. A continuación, normalizamos los valores de los gradientes para no tener el problema de *explosión de gradientes* y prevenir la inestabilidad numérica que puede acarrear. Por último, actualizamos el *optimizador* para mejorar los valores de los parámetros del modelo según los *gradientes* de pérdida y actualizamos el *scheduler* para ajustar la tasa de aprendizaje del *optimizador*. El código utilizado para este proceso se muestra en la Figura 4.40.

Para mejorar el resultado del modelo decidimos implementar el entrenamiento del modelo por épocas de manera que se entrena y evalúa un modelo con nuestros *TensorDataset* de entrenamiento y validación completos para cada época. Esta implementación permite que el modelo se entrene a partir de un modelo ya entrenado con nuestro *TensorDataset* para obtener una mejora incremental en el rendimiento. Además, para evitar recorrer el número de épocas completo en caso de no ser necesario, implementamos lógica de *parada temprana*. Esta lógica nos permite parar la iteración cuando no ha habido una mejora del modelo en un número de épocas determinado a voluntad.

```
model.train()
total_loss = 0

for step, batch in enumerate(train_loader):
    batch = [t.to(device) for t in batch]
    b_input_ids, b_input_mask, b_labels = batch

    model.zero_grad()
    outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)
    loss = outputs.loss
    total_loss += loss.item()
    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
    optimizer.step()
    scheduler.step()
```

Figura 4.40: Código para el entrenamiento del modelo por lotes

Tras adaptar la función *evaluate* para que devuelva también las predicciones y las etiquetas podremos mostrar los resultados del mejor modelo adecuadament (Figura 4.42). Esta nueva función es utilizada en el código de la Figura 4.43 con el cual mostramos que el mejor modelo se ha cargado correctamente e imprimimos su pérdida y exactitud de validación así como su Matriz de Confusión para comprobar que efectivamente es el modelo correcto. Después mostramos los resultados obtenidos de la nueva función *evaluateBestModel* en forma de informe de métricas y matriz de confusión que veremos en el siguiente capítulo.

```python
# Entrenamiento
# Inicializar variables para la parada temprana
best_val_accuracy = 0.0
best_val_loss = float('inf')  # Añadido para rastrear la mejor pérdida de validación
patience = 3  # Número de épocas para esperar una mejora antes de detener el entrenamiento
patience_counter = 0
# Inicializar variable para guardar el mejor modelo
best_model_state = None
best_epoch = 0

for epoch in range(epochs):
    model.train()
    total_loss = 0

    for step, batch in enumerate(train_loader):
        batch = [t.to(device) for t in batch]
        b_input_ids, b_input_mask, b_labels = batch

        model.zero_grad()
        outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

    avg_train_loss = total_loss / len(train_loader)
    print(f"Epoch {epoch + 1}/{epochs} - Training Loss: {avg_train_loss}")

    # Evaluación después de cada época
    val_accuracy, val_loss = evaluate(model, val_loader)
    print(f"Validation Accuracy: {val_accuracy}")
    print(f"Validation Loss: {val_loss}")

    # Lógica de parada temprana basada en la precisión
    if val_accuracy > best_val_accuracy :
            best_val_accuracy = val_accuracy
            patience_counter = 0
            best_model_state = copy.deepcopy(model.state_dict())
            best_epoch = epoch
            print(f"New best model found at epoch {epoch+1}")
    else:
        patience_counter += 1
        if patience_counter >= patience:
            print(f"Parada temprana: La precisión de validación no ha mejorado en las últimas {patience} épocas.")
            break
```

Figura 4.41: Código de el entrenamiento y la evaluación por épocas

```python
def evaluateBestModel(model, val_loader):
    model.eval()
    total_eval_loss = 0
    predictions, true_labels = [], []

    for batch in val_loader:
        b_input_ids, b_input_mask, b_labels = [t.to(device) for t in batch]
        with torch.no_grad():
            outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)

        total_eval_loss += outputs.loss.item()
        logits = outputs.logits
        probs = softmax(logits.detach().cpu().numpy(), axis=1)
        pred_labels = np.argmax(probs, axis=1)
        label_ids = b_labels.to('cpu').numpy()

        predictions.extend(pred_labels)
        true_labels.extend(label_ids.flatten())

    avg_val_accuracy = accuracy_score(true_labels, predictions)
    avg_val_loss = total_eval_loss / len(val_loader)

    return avg_val_accuracy, avg_val_loss, predictions, true_labels
```

Figura 4.42: Código de la función *evaluateBestModel*

```python
# Carga el mejor estado del modelo
if best_model_state:
    model.load_state_dict(best_model_state)
    print("Mejor modelo cargado")

    # Evalúa el modelo y obtiene las predicciones y las etiquetas verdaderas
    val_accuracy, val_loss, predictions, true_labels = evaluateBestModel(model, val_loader)

    print("Validation Loss: ", val_loss)
    print("Validation Accuracy: ", val_accuracy)

    # Calcula la matriz de confusión
    conf_matrix = confusion_matrix(true_labels, predictions)

    # Etiquetas personalizadas para las clases
    class_labels = ['Negative', 'Neutral', 'Positive']

    # Grafica la matriz de confusión como un mapa de calor
    plt.figure(figsize=(10, 7))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
    plt.title('Matriz de Confusión')
    plt.xlabel('Predicho')
    plt.ylabel('Verdadero')
    plt.show()

    # Imprime el informe de clasificación
    print(classification_report(true_labels, predictions, target_names=class_labels, zero_division=0))
```

Figura 4.43: Código para dividir *dataset* de entrenamiento y evaluación

Módulo	**Librería**	**Descripción**
torch	torch	Proporciona tensores multidimensionales y operaciones de álgebra lineal, optimización, etc.
DataLoader, TensorDataset, random_split	torch.utils.data	Facilita la carga de datos y su manejo en PyTorch.
AdamW	torch.optim	Implementación del optimizador Adam con corrección de decaimiento del peso.
BertForSequenceClassification, BertTokenizer	transformers	Facilita el uso de modelos BERT para clasificación de secuencias y tokenización.
get_linear_schedule_with_warmup	transformers	Tasa de aprendizaje con un periodo de calentamiento lineal.
accuracy_score, classification_report, confusion_matrix	sklearn.metrics	Métricas para evaluar el rendimiento del modelo.
softmax	scipy.special	Proporciona la función softmax.
matplotlib.pyplot	matplotlib	Utilizado para crear gráficos estáticos, animados e interactivos.
seaborn	seaborn	Basado en matplotlib, ofrece una interfaz de alto nivel para dibujar gráficos estadísticos atractivos y informativos.
pandas	pandas	Proporciona estructuras de datos y herramientas para la manipulación y análisis eficientes de datos.
numpy	numpy	Librería fundamental para la computación científica en Python.
copy	copy	Utilizado para realizar copias superficiales y profundas de objetos.

Tabla 4.6: Módulos utilizados en la implementación del modelo BERT

Capítulo 5

Resultados

5.1. Métricas para el Análisis

En esta sección vamos a describir las distintas métricas que hemos utilizado para evaluar los modelos, así como su significado y su utilidad concreta para nuestro caso [43].

5.1.1. Introducción

La validación de los modelos es un proceso crítico que garantiza la evaluación de la calidad y la fiabilidad de los mismos. Las métricas de evaluación son, por lo tanto, herramientas esenciales que cuantifican el rendimiento del modelo bajo ciertos criterios estadísticos.

El propósito de esta sección es ofrecer una visión clara de las métricas de evaluación más relevantes que utilizaremos como conjunto de herramientas para juzgar objetivamente la eficacia de cada modelo desarrollado. Es crucial considerar múltiples métricas para obtener una perspectiva integral del comportamiento del modelo, evitando posibles interpretaciones sesgadas que pueden surgir del uso de una sola métrica de rendimiento. En contextos donde las clasificaciones son no binarias, esto es, involucran múltiples clases - en nuestro caso, positivo, negativo y neutro - la matriz de confusión se extiende a más de dos dimensiones para capturar la complejidad del problema. Esta matriz sigue siendo fundamental para diagnosticar el comportamiento del modelo de clasificación, pero ahora se analizan tres tipos de resultados correctos e incorrectos para cada clase:

5.1.2. Matriz de Confusión (*Confusion Matrix*)

La matriz de confusión es una herramienta valiosa para entender el comportamiento de un modelo en términos de sus aciertos y errores. Representa las clasificaciones correctas e incorrectas de manera tabular, diferenciando entre las clases positivas y negativas, así como las predicciones del modelo. Los elementos de esta matriz se definen como:

- Verdaderos Positivos (VP): Hace referencia a los casos donde la clase positiva ha sido correctamente identificada.
- Verdaderos Negativos (VN): Hace referencia a los casos donde la clase negativa ha sido correctamente identificada.
- Falsos Positivos (FP): Hace referencia a los casos donde la clase negativa ha sido incorrectamente identificada como positiva.
- Falsos Negativos (FN): Hace referencia a los casos donde la clase positiva ha sido incorrectamente identificada como negativa.

Para cada clase, se consideran como verdaderos negativos todas aquellas instancias que el modelo ha clasificado correctamente en las otras clases.

5.1.3. Precisión (*Precision*)

La Precisión mide la proporción de predicciones positivas que son realmente correctas. Es decir, la precisión es la habilidad del modelo para no clasificar como positivo un caso negativo. Es importante tenerlo en cuenta, sobretodo cuando la cantidad de Falsos Positivos es elevada. Se calcula como:

$$Precision = VP/(VP + FP)$$

5.1.4. Sensibilidad (*Recall*)

La Sensibilidad mide la capacidad del modelo para identificar correctamente las instancias positivas. Es crucial cuando es importante detectar todos los casos positivos. Se calcula como:

$$Recall = VP/(VP + FN)$$

5.1.5. Puntuación F1 (*F1-Score*)

La Puntuación F1 combina la Precisión y la Sensibilidad en una sola métrica, proporcionando un balance entre ambas. El Puntuación F1 es la media harmónica de estas dos métricas. Al igual que la Precisión y la Sensibilidad su valor ideal es 1 y el peor caso es 0. Es especialmente útil en situaciones donde se busca una medida que tome en cuenta tanto la capacidad del modelo para identificar correctamente las clases positivas como para evitar falsos positivos.

$$F1Score = 2 * (Precision * Recall)/(Precision + Recall)$$

5.1.6. Exactitud (*Accuracy*)

La Exactitud [44] refleja el porcentaje total de predicciones correctas y es útil como una medida general del rendimiento. Se debe usar con precaución, particularmente cuando las clases están desequilibradas. Se calcula como el porcentaje de

predicciones correctas hechas por el modelo sobre el total de predicciones realizadas:

$$Accuracy = (VP + VN)/(VP + VN + FP + FN)$$

5.1.7. Promedios (Macro Average and Weighted Average)

El Promedio Macro calcula la media de las métricas para cada clase sin tener en cuenta ni el ponderado ni el soporte, mientras que el Promedio Ponderado toma en cuenta el soporte de cada clase, proporcionando una medida que refleja la distribución de clases en el conjunto de datos [45]. El Promedio Macro ofrece una visión equitativa del rendimiento al calcular la media aritmética de las métricas de cada clase, sin dar más importancia a ninguna clase en particular, lo que es útil para evaluar el rendimiento del modelo cuando las clases no están balanceadas. Por otro lado, el Promedio Ponderado ajusta la contribución de cada clase a la media general en función de su presencia en los datos, lo cual es crucial para reflejar la influencia de cada clase en la precisión general del modelo debido a su frecuencia relativa en el conjunto de datos.

5.1.8. Support

La métrica support hace referencia al tamaño del conjunto de datos de entrenamiento, es decir, el número de comentarios que se han utilizado para entrenar el modelo

5.2. K-Nearest Neighbours

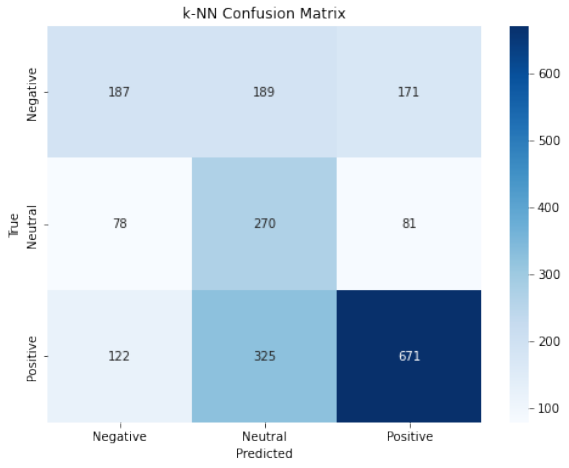

Figura 5.1: Matriz de confusión del KNN

Los resultados del modelo K-Nearest Neighbours, que se pueden ver en la Figura 5.2, la Tabla 5.2 y la Tabla 5.1 y la matriz de confusión apropiada se puede ver en la Figura 5.1, nos indican que su comportamiento no es idóneo al tener una precisión del 54 %. Si bien el porcentaje de aciertos es mayor que al clasificar aleatoriamente,

Clase	Precisión	Exhaustividad	Punt. F1	Exactitud	Support
Negativo	0,48	0,34	0,4	0.54	547
Neutral	0,34	0,63	0,45		429
Positivo	0,73	0,60	0,66		1118

Tabla 5.1: Métricas de clasificación K Nearest Neighbours

Métrica Average	Precisión	Exhaustividad	Puntuación F1
Macro Avg.	0,52	0,52	0,50
Weighted Avg.	0,58	0,54	0,55

Tabla 5.2: Métricas de clasificación K-Nearest Neighbours

```
Accuracy: 0.54
              precision    recall  f1-score   support

    Negative       0.48      0.34      0.40       547
     Neutral       0.34      0.63      0.45       429
    Positive       0.73      0.60      0.66      1118

    accuracy                           0.54      2094
   macro avg       0.52      0.52      0.50      2094
weighted avg       0.58      0.54      0.55      2094
```

Figura 5.2: Resultados del KNN

aún existe un margen de mejora. El modelo se comporta correctamente a la hora de clasificar comentarios positivos con una precisión del 73 % y una sensibilidad del 60 %. Sin embargo, el modelo tiene claros problemas a la hora de clasificar comentarios negativos como se puede ver con su precisión del 48 % y sensibilidad del 34 %. Los comentarios neutrales dan problemas en la precisión con un 34 %, no obstante, tienen una sensibilidad aceptable, 63 %, lo cual indica que en este caso tienen una gran capacidad de acierto. Hay varias causas para explicar el pobre resultado de este modelo, teniendo en cuenta que cada palabra se corresponde con una variable, lo cual crea una gran dimensionalidad, una posible causa es la conocida como *maldición de la dimensión* [46]. Esto indica que en espacios de gran dimensionalidad, la noción de proximidad se diluye debido a la dispersión de los datos y a la convergencia de las distancias cuando hay un gran número de dimensiones [47].

5.3. Multinomial Naive Bayes

Como se puede observar en los resultados, Figura 5.3, Tabla 5.3 y Tabla 5.4, el modelo Multinomial Naive Bayes se comporta relativamente bien identificando comentarios de sentimiento positivo, obteniendo una precisión del 65 % y una sensibilidad del 91 %. Con los comentarios negativos el comportamiento no es excesivamente bueno pero podría considerarse aceptable con una precisión del 65 % y una sensibilidad del 50 %. Sin embargo, la clase neutral da muchos problemas ya que el

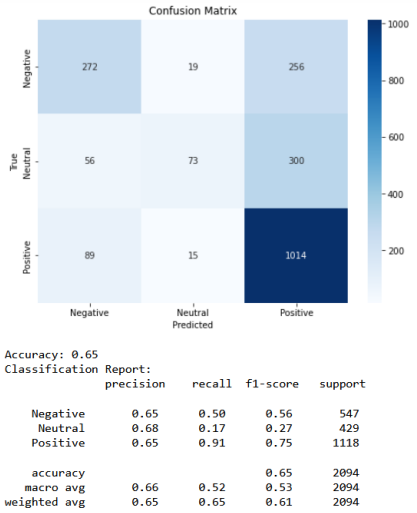

Figura 5.3: Matriz de confusión del Multinomial Naive Bayes

modelo no es capaz de discernir correctamente los comentarios neutrales como se puede apreciar en la baja sensibilidad de esta clase que es del 17 % y en la Puntuación F1 que es del 27 %. En este caso eliminar la clase neutral podría ser beneficioso para el análisis empleando este modelo.

Clase	Precisión	Exhaustividad	Punt. F1	Exactitud	Support
Negativo	0,65	0,50	0,56	0.65	547
Neutral	0,68	0,17	0,27		429
Positivo	0,65	0,91	0,75		1118

Tabla 5.3: Métricas de clasificación Multinomial Naive Bayes

Métrica average	Precisión	Exhaustividad	Puntuación F1
Macro Avg.	0,66	0,52	0,53
Weighted Avg.	0,65	0,65	0,61

Tabla 5.4: Promedios de métricas de clasificación Multinomial Naive Bayes

5.4. Random Forest

A continuación profundizaremos un poco sobre las métricas proporcionadas tras el entrenamiento del modelo cuyo resultado se recoge en la Tabla 5.5. Sorprende que a pesar de haber utilizado dos tipos de vectorizaciones distintas (Count Vectorizer y Tf-Idf), los resultados son bastante similares, siendo el modelo en que se usa Count

Vectorizer ligeramente mejor, con una precisión general del 74 % frente a un 73 % del modelo con la vectorización Tf-Idf. Esto significa que en el futuro, si utilizamos este modelo, lo haremos con Count Vectorizer. En ambos casos podemos observar que el rendimiento del modelo en la clase negativa es bastante pobre, siendo la puntuación F1 53 % con Count Vectorizer y de 49 % con Tf-Idf. Se podría pensar que se debe a un desbalance de clases, y en parte es cierto, pero si observamos la clase neutra, su conjunto de prueba es incluso menor y se obtiene un resultado bastante optimista, con unas métricas rondando entre el 68 % y el 76 %. Por tanto, podemos concluir que la clase neutra tiene características más representativas que la negativa, a pesar de que cabría esperarse lo contrario. Esto se puede deber a que nuestro modelo tiene un sesgo sobre la clase negativa que no podemos determinar claramente, o que la clase negativa tenga más variabilidad en sus comentarios, haciéndola más difícil de clasificar.

En cuanto a la clase positiva, en ambas vectorizaciones hemos obtenido una cantidad de verdaderos positivos muy alta. De 1118 comentarios positivos, en las dos vectorizaciones hemos obtenido mas de 1000 verdaderos positivos, lo que significa que nuestro modelo clasifica los comentarios positivos de forma correcta (solo aproximadamente un 10 % de los comentarios positivos se han clasificado incorrectamente). Sin embargo, cabe recalcar que entre 320 y 350 comentarios han sido clasificados como Positivos cuando en realidad eran neutros o negativos. Esto ocurre sobretodo en la clase Negativa, y los motivos por los que esto puede ocurrir se detallan en el párrafo anterior.

Clase	Precisión	Exhaustividad	Punt. F1	Exactitud	Support
Negativo	0,76	0,40	0,53	0.74	547
Neutral	0,68	0,76	0,72		429
Positivo	0,76	0,90	0,82		1118

Tabla 5.5: Métricas de clasificación Random Forest con CountVectorizer

También observamos el rendimiento pobre en la clasificación de comentarios Negativos al observar los resultados obtenidos para los promedios de las métricas mostrados en la Tabla 5.6. Se denota en la diferencia que hay entre el promedio sin tener en cuenta los pesos de las clases y el promedio con pesos.

El resultado de la ejecución del modelo con Count Vectorizer a partir del que obtenemos estos resultados se observa en la Figura 4.18 y la matriz de confusión en la Figura 5.4.

Los resultados con TF-IDF también se pueden observar en las Tablas 5.7 y 5.8, mientras que la matriz de confusión se puede observar en la Figura 5.5.

En definitiva, podemos determinar que este modelo funciona correctamente con las clases positiva y negativa, pero tiene un rendimiento bastante bajo con la clase negativa, siendo este modelo muy poco útil para clasificar comentarios de dicho tipo.

Métrica average	Precisión	Exhaustividad	Puntuación F1
Macro Avg.	0,73	0,69	0,69
Weighted Avg.	0,74	0,74	0,72

Tabla 5.6: Métricas average de Random Forest con Count Vectorizer

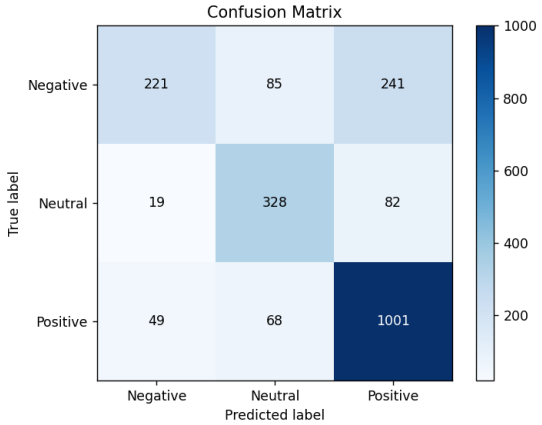

Figura 5.4: Matriz de confusión del Random Forest con Count Vectorizer

Clase	Precisión	Exhaustividad	Punt. F1	Exactitud	Support
Negativo	0,80	0,35	0,49	0.73	547
Neutral	0,68	0,77	0,72		429
Positivo	0,74	0,91	0,81		1118

Tabla 5.7: Métricas de clasificación Random Forest con Tf-Idf

Métrica Average	Precisión	Exhaustividad	Puntuación F1
Macro Avg.	0,74	0,67	0,67
Weighted Avg.	0,74	0,73	0,71

Tabla 5.8: Métricas average para Random Forest con Tf-Idf

5.5. Regresión Logística Multinomial

A continuación se reflejan los resultados proporcionados por el clasificador con regresión logística.

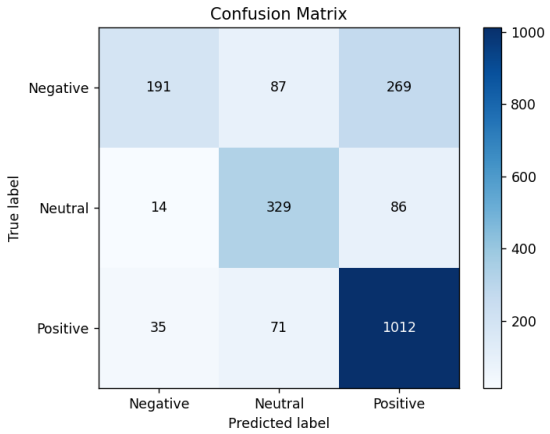

Figura 5.5: Matriz de confusión de Random Forest con Tf-Idf

Métricas de Regresión Logística

Una vez entrenado el modelo podemos ver las métricas relacionadas con su rendimiento. Si miramos la Tabla 5.9, observamos que el rendimiento del modelo con la clase negativa (63 % puntuación F1) es ligeramente inferior que para la clase neutral (69 % puntuación F1) y significativamente menor que para la clase positiva, lo cual tiene sentido porque la mayoría de los comentarios de nuestro *dataset* son positivos (dando lugar a un conjunto poco balanceado). Es por esto por lo que nuestros modelos van a aprender mucho mejor de la clase positiva que del resto de clases. Podemos observar que el funcionamiento del modelo para esta clase es bastante notable, teniendo una puntuación F1 del 82 %, que es bastante buena. Podemos observar también en la Tabla 5.10 que las métricas promedio son bastante buenas, con 0,75 para el promedio ponderado de la puntuación F1.

En la Figura 5.6 observar que se han clasificado 925 comentarios positivos reales como positivos, mientras que 109 han sido clasificados como negativos y 84 como neutrales. De todos los comentarios clasificados como positivos, realmente 925 eran realmente positivos, mientras que 73 realmente eran neutrales y 129 eran negativos. Para las clases neutrales y negativa tenemos un rendimiento mas o menos similar teniendo 321 comentarios neutros como realmente neutros y 318 clasificados como negativos siendo negativos.

Podemos concluir que este modelo funciona satisfactoriamente, con una precisión general del 75 %. Si queremos mejorar el rendimiento en las clases neutral y negativo, tendríamos que facilitar a los modelos conjuntos de datos más equilibrados, pero nuestros datos han sido extraídos de un conjunto de datos real, con lo cual también queríamos reflejar el sentimiento general de los comentarios extraídos.

5.6. SVM

A continuación se reflejan los resultados proporcionados por el clasificador con SVM.

Clase	Precisión	Exhaustividad	Punt. F1	Exactitud	Support
Negativo	0,69	0,58	0,63	0.75	547
Neutral	0,64	0,75	0,69		429
Positivo	0,82	0,83	0,82		1118

Tabla 5.9: Métricas de clasificación Regresión Logística

Métrica average	Precisión	Exhaustividad	Puntuación F1
Macro Avg.	0,71	0,72	0,71
Weighted Avg.	0,75	0,75	0,75

Tabla 5.10: Métricas average de Regresión Logística

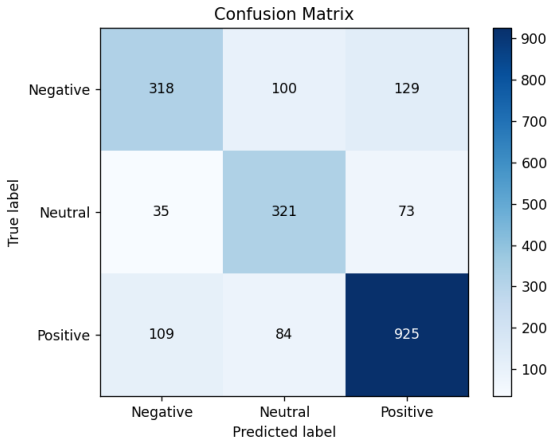

Figura 5.6: Matriz de confusión de la regresión logística

5.6.1. SVC

La Figura 5.7 muestra el informe de los valores obtenidos para nuestras métricas de análisis y promedios en el caso del código para el Modelo SVM con la estrategia OvO.

	precision	recall	f1-score	support
Negative	0.71	0.57	0.63	811
Neutral	0.69	0.60	0.64	633
Positive	0.77	0.89	0.83	1696
accuracy			0.75	3140
macro avg	0.73	0.69	0.70	3140
weighted avg	0.74	0.75	0.74	3140

Figura 5.7: Informe de métricas del modelo SVM con SVC

Los resultados obtenidos para las métricas de análisis se recogen en la Tabla 5.11. Podemos ver que el modelo *SVC* tiene una exactitud media del 75 % con un

valor de F1 máximo perteneciente a los comentarios positivos 83 % y el mínimo que se corresponde a los negativos 63 % aunque muy cercano al de los comentarios etiquetados como neutros: 64 %. Sin embargo, hay una diferencia de un 16 % entre la precisión y la sensibilidad, a favor de la primera, en la clasificación de comentarios negativos y de un 9 % para los comentarios neutros. Esto indica que el modelo es algo conservador a la hora de predecir los comentarios negativos, ya que, aunque la mayoría de predicciones son correctas, un número considerable de observaciones negativas no se clasifican correctamente.

Clase	Precisión	Exhaustividad	Punt. F1	Exactitud	Support
Negative	0,71	0,57	0,63	0.75	811
Neutral	0,69	0,60	0,64		633
Positive	0,77	0,89	0,83		1696

Tabla 5.11: Métricas de clasificación del modelo SVM con SVC

En cuanto a los promedios de las métricas se denota que el valor del promedio sin considerar pesos se ve afectado por el peor rendimiento del modelo en la clasificación de las clases minoritarias como refleja la Tabla 5.12

Métrica average	Precisión	Exhaustividad	Puntuación F1
Macro Avg.	0,73	0,69	0,70
Weighted Avg.	0,74	0,75	0,74

Tabla 5.12: Métricas globales del modelo SVM con SVC

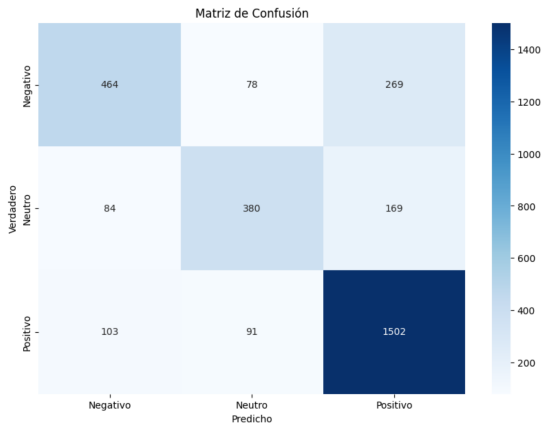

Figura 5.8: Matriz de confusión del modelo SVM con SVC

La matriz de confusión de la Figura 5.8 muestra una alta densidad en los casos *VP* para la clase positiva y una densidad menor para los *VP* del resto de clases aunque también se aprecie una densidad mayor de estas respecto al resto de casos.

No obstante la densidad de casos predichos como positivos que resultan ser realmente negativos es considerablemente alta respecto al resto de casos de *FP*.

5.6.2. OvR

La Figura 5.9 muestra el informe de los valores obtenidos para nuestras métricas de análisis en el caso del Modelo *SVM* con la estrategia *OvR*.

	precision	recall	f1-score	support
Negative	0.70	0.61	0.65	811
Neutral	0.70	0.67	0.68	633
Positive	0.80	0.86	0.83	1696
accuracy			0.76	3140
macro avg	0.73	0.72	0.72	3140
weighted avg	0.76	0.76	0.76	3140

Figura 5.9: Informe de las métricas para el modelo SVM con OvR

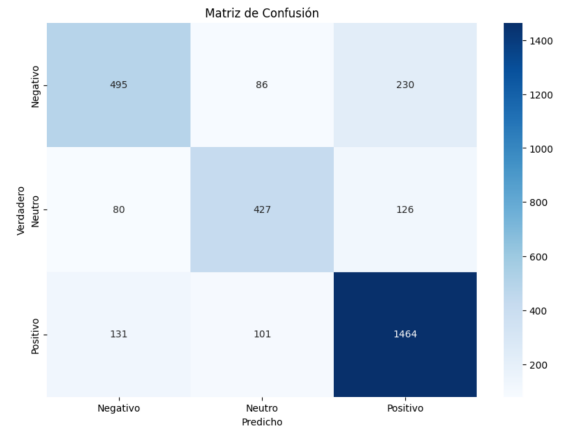

Figura 5.10: Matriz de confusión del Modelo SVM con OvR

Los resultados que ofrece el modelo *SVM* con enfoque *OvR*, recogidos en la Tabla 5.13, son muy parecidos a los del modelo *SVC* con una exactitud del 76 %, tan solo un 1 % mayor. No obstante, podemos denotar que los valores del recall son un 4 % y 7 % mayores para los casos negativos (61 %) y neutros (67 %) respectivamente pero un 3 % menor para los casos positivos (83 %). En general se consiguen unos resultados más balanceados y se puede concluir que el modelo es mejor para identificar cada caso como tal pese a que la exactitud total no se ve afectada notablemente.

Los resultados de los promedios de las métricas de análisis para el modelo SVM con enfoque *OvR* dados en la Tabla 5.14 que muestra un resultado parecido al anterior modelo SVM pero con menor diferencia entre ambos dada la mejora en las métricas de clasificación de las clases minoritarias.

La matriz de confusión reflejada en la Figura 5.10 muestra una ligera mejora en la densidad de los casos *VP* para la clase neutral y una menor densidad respecto al modelo SVC para los casos predichos como positivos siendo negativos realmente. En general observamos una ligera mejora al implementar el enfoque *OneVsRest* para

Clase	Precisión	Exhaustividad	Punt. F1	Exactitud	Support
Negativo	0,70	0,61	0,65	0.76	811
Neutral	0,70	0,67	0,68		633
Positivo	0,80	0,86	0,83		1696

Tabla 5.13: Métricas de clasificación por clase

Métrica average	Precisión	Exhaustividad	Puntuación F1
Macro Avg.	0,73	0,72	0,72
Weighted Avg.	0,76	0,76	0,76

Tabla 5.14: Métricas promedio

el modelo SVC sobretodo en las métricas representativas de las clases minoritarias (*Negativo* y *Neutral*)

5.7. BERT

Para el análisis de resultados de BERT tomamos el informe de las métricas y la matriz de confusión del mejor estado del modelo como muestran las Figuras 5.11 y 5.12 respectivamente.

	precision	recall	f1-score	support
Negative	0.79	0.84	0.82	563
Neutral	0.89	0.78	0.83	444
Positive	0.89	0.90	0.89	1087
accuracy			0.86	2094
macro avg	0.86	0.84	0.85	2094
weighted avg	0.86	0.86	0.86	2094

Figura 5.11: Informe de las métricas para el modelo BERT

Los resultados del informe se recogen en las Tablas 5.15 y 5.16.

Clase	Precisión	Exhaustividad	Punt. F1	Exactitud	Support
Negativo	0,79	0,84	0,82	0.86	563
Neutral	0,89	0,78	0,83		444
Positivo	0,89	0,90	0,89		1087

Tabla 5.15: Métricas de clasificación por clase para el modelo BERT

Como podemos observar en la Tabla 5.15 el modelo tiene un exactitud del 86 %, bastante óptima. También apreciamos un desnivel entre los valores del recall para

las distintas clases. La clase positiva tiene un recall del 90 % mientras que para la neutral y negativa es del 78 % y 84 %. Esto sugiere que el modelo es especialmente eficaz identificando los comentarios positivos. La diferencia entre el valor de los comentarios positivos frente al resto puede deberse al menor volumen de muestras de comentarios negativos y neutros, lo que implica una mayor dificultad para el modelo a la hora de identificar estas clases.

En cuanto a la precisión podemos ver unos valores del 89 % para comentarios predichos positivos, 89 % para neutros y 79 % para negativos. La mayor diferencia entre el valor de precisión y exhaustividad la hallamos en la clase neutral. Esto significa que el modelo suele acertar cuando clasifica un comentario en la clase negativa pero también hay un número considerable de casos que pertenecen realmente a esta clase y el modelo no los clasifica como tal. Es decir, el modelo es más conservador a la hora de clasificar un comentario como neutro. Por otra parte, observamos que el comportamiento contrario con la clase negativa ya que dada su baja precisión y su elevada exhaustividad respecto a la clase neutral indica que el modelo tiende a clasificar comentarios en este clase pese a que no lo son para asegurarse de identificar correctamente la mayor cantidad de comentarios de esta clase aunque también clasifique comentarios de otras clases como negativos. Esto puede deberse a que en nuestro conjunto de datos no hay una caracterización lingüística marcada claramente para esta clase.

No obstante, si observamos la puntuación F1 veremos que se nivelan los valores, un 83 % y 82 % para las clases neutral y negativa. Esto indica que el rendimiento para detectar esta clases es decente pero dado el estudio hecho de la diferencia entre la precisión y la exhaustividad para estas clases sabemos que en cada caso tiene un comportamiento especifico más marcado. Al final se puede concluir que el modelo es muy preciso al identificar casos neutrales solo en caso de serlo y los casos negativos aunque es más probable que clasifique un comentario que no corresponde a esta clase como tal en este último caso. No obstante, el modelo destaca con creces en la clasificación de comentarios positivos dados los elevados valores de sus métricas y el equilibrio entre estas.

Métrica average	**Precisión**	**Exhaustividad**	**Puntuación F1**
Macro Avg.	0,86	0,84	0,85
Weighted Avg.	0,86	0,86	0,86

Tabla 5.16: Métricas promedio para el modelo BERT

En cuanto a los resultados de los promedios de las métricas mostrados en la tabla 5.16, hay una diferencia de un 2 % o menor a favor del promedio pesos. Resulta una diferencia mínima y despreciable dado que la clase positiva que es la que mejor clasifica el modelo es también la clase con mayor número de muestras utilizadas para el entrenamiento y clasificación. Por tanto, podemos afirmar que nuestro modelo tiene buena precisión para clasificar todas las clases.

Si observamos la matriz de confusión en la Figura 5.12 vemos que los recuadros con mayor densidad son los casos de VP para cada clase. Con un breve análisis de

Figura 5.12: Matriz de confusión del Modelo BERT

los resultados se puede afirmar que el modelo es bastante eficaz en la clasificación de los comentarios en su respectiva clase aunque, como el resto, está ligeramente mejor enfocado en la clasificación de comentarios positivos.

Capítulo 6

Análisis de Resultados

En este capítulo realizaremos un análisis más profundo de los resultados obtenidos y de la pertinencia con nuestro tema de estudio. Para ello tomaremos el modelo con mejor rendimiento observado en el capitulo anterior.

6.1. Análisis de los resultados del modelo BERT

Una vez vistos los rendimientos de los distintos modelos entrenados optamos por realizar un análisis profundo de los resultados obtenidos con el modelo BERT dado que este ha sido sin duda el que mejor resultados nos ha dado. Para analizar cómo ha clasificado el modelo los comentarios empezaremos extrayendo las palabras que más ha tenido en cuenta en el proceso de clasificación. Esta tarea no es trivial cuando se trabaja con modelos BERT y a día de hoy la interpretación de resultados de modelos BERT sigue siendo un área de investigación activa. No obstante, existen técnicas para realizar esta tarea de interpretación como Integrated Gradients [48; 49] (IG) de Captum [50]. A continuación explicaremos en que consiste esta técnica.

6.1.1. Integrated Gradients

El objetivo principal de la técnica de explicación Integrated Gradients es la interpretación de la toma de decisiones de modelos como BERT. IG funciona cuantificando la importancia relativa de los token de entrada en el proceso de clasificación del modelo en un valor conocido como *atribución*. El cálculo de esta valor se realiza a partir del cálculo de la integral de los *gradientes* desde un *punto de referencia* o *baseline*(comentario neutral de ejemplo sobre el tema general del *dataset*) hasta la *entrada de interés* (comentario a interpretar). Calcula los gradientes de la salida del modelo respecto a la entrada y se integran a lo largo de la distancia en el espacio de entrada entre el *punto de referencia* y la *entrada de interés*. El valor obtenido en esta integración reflejará la importancia de cada token de entrada en la predicción realizada por el modelo.

Para implementar esta técnica utilizaremos la biblioteca Integrated Gradients de Captum. Primero preparamos una pequeña muestra de nuestro *dataset* con la que trabajar. Declaramos la función *forward_func* que alimentará las salidas del

modelo a la función IntegratedGradients. Después procesamos la muestra por lotes y preparamos el *baseline* con la frase *"El tema es OpenAI"* para pasársela a la función *attribute* de *IG* como se muestra en la figura 6.1. Durante la implementación nos topamos con varios errores de difícil resolución. El mayor problema fue conocer la manera en la que Captum interactúa con BERT en el manejo de los datos como se puede ver en la figura 6.2. Para resolver este problema adaptamos la función *forward_func* para asegurarnos de que el tipo de los datos que devuelve son *Tensores* [51] de tipo *Long* como refleja la figura 6.3.

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from captum.attr import IntegratedGradients

# Preparar una muestra del conjunto de validación para el análisis
sample_size = 10  # Ajustamos el tamaño de la muestra
sample_dataset, _ = random_split(val_dataset, [sample_size, len(val_dataset) - sample_size])
sample_loader = DataLoader(sample_dataset, batch_size=1)

# Función de forward
def forward_func(input_ids, attention_mask=None):
    return model(input_ids, attention_mask=attention_mask)[0]

# Integrated Gradients
ig = IntegratedGradients(forward_func)

# Bucle para procesar el conjunto de datos de muestra
for batch in sample_loader:
    b_input_ids, b_attention_mask, _ = [t.to(device) for t in batch]

    # Baseline (usando una frase específica)
    baseline_text = "El tema es OpenAI."
    baseline_ids = tokenizer.encode(baseline_text, return_tensors='pt', max_length=128, truncation=True, padding='max_length'
    baseline_mask = torch.zeros_like(baseline_ids).to(device)

    # Calcular atribuciones
    attributions = ig.attribute(inputs=(b_input_ids, b_attention_mask),
                                baselines=(baseline_ids, baseline_mask))
```

Figura 6.1: Código inicial para la interpretación del modelo BERT con IG de Captum

```
RuntimeError: Expected tensor for argument #1 'indices' to have one of the following scalar types: Long, Int; but got torch.
cuda.FloatTensor instead (while checking arguments for embedding)
```

Figura 6.2: Salida del código de la figura 6.1

```
# Función de forward
def forward_func(input_ids, attention_mask=None):
    if attention_mask is None:
        attention_mask = torch.ones_like(input_ids)

    # Obtener los logits del modelo
    logits = model(input_ids.long(), attention_mask=attention_mask.long())[0]

    # Seleccionar el logit de la clase con la puntuación más alta (o una clase específica)
    return torch.max(logits, dim=1).values
```

Figura 6.3: Función de forward de la figura 6.1 adaptada

```
RuntimeError: One of the differentiated Tensors appears to not have been used in the graph. Set allow_unused=True if this is
the desired behavior.
```

Figura 6.4: Salida del código con la función forward de la figura 6.3 adaptada

```
# Calcular atribuciones
attributions = ig.attribute(inputs=(b_input_ids, b_attention_mask),
                            baselines=(baseline_ids, baseline_mask),allow_unused=True)
```

Figura 6.5: Llamada a la función attribute adaptada para resolver el error de la figura 6.4

```
TypeError: attribute() got an unexpected keyword argument 'allow_unused'
```

Figura 6.6: Salida para la llamada a attribute de la figura 6.5

Tras resolver el problema anterior nos vimos encerrados en un punto muerto como muestran las Figuras 6.4, 6.5 y 6.6 del que no fuimos capaces de salir.

Dada la escasez de documentación y de trabajos publicados en este área a parte de la alta dificultad de la naturaleza de la tarea a desarrollar nos vimos obligados a buscar una solución alternativa.

6.1.2. Mecanismo de Atención

El mecanismo de atención en modelos como BERT es la técnica que permite a estos modelos ponderar la importancia de las relaciones entre las palabras de una secuencia de texto. Esta función calcula una puntuación de *atención* para cada par de palabras la cual refleja la importancia de la palabra puntuada respecto al resto de palabras de la secuencia de entrada.

Puesto que no logramos utilizar correctamente Integrated Gradients probamos a obtener las palabras más importantes según el valor de la atención de nuestro modelo. Obtendremos un diccionario con cada token junto a su atención para cada clase *important_words* como se muestra en la Figura 6.7. Para mostrar los resultados iteramos por el diccionario mostrando las 10 palabras con mayor recuento para cada clase y el propio recuento como muestra la Figura 6.8.

Al obtener el diccionario de palabras importantes observamos que hay muchas palabras que coinciden entre clases ya que el mecanismo de atención no mide la importancia del token en la clasificación si no que mide que palabras tienen más peso en cuanto al significado del comentario y la importancia relativa en su contexto. Por tanto, estas palabras resultan clave para que el modelo comprenda el contexto y el tema principal de los comentarios como se puede apreciar por las palabras mostradas. Dado que las palabras obtenidas se entienden como contextuales y no tan características de la decisión para los distintos comentarios decidimos obtener las palabras más importantes exclusivas de cada clase, es decir, aquellas que no aparecen en el resto de clases, con la intención de obtener una visión más realista de las palabras que pueden tener más importancia en términos generales a la hora de clasificar un comentario en cada clase en concreto. La función *get_exclusive_important_words* mostrada en la Figura 6.9 realiza la extracción de las palabras exclusivas por sentimiento del diccionario *important_words*.

Utilizamos la biblioteca *Seaborn* [52] para representar gráficamente las 10 palabras que más aparecen exclusivas a cada clase.

Para la clase negativo observamos en la Figura 6.10 palabras interesantes como *terrorist*,*destroyer*,*accident*, *outlaw*, *nightmare* y *painfully* que pueden entenderse como palabras con un sentimiento característico de esta clase que se entiende que pertenecen a comentarios que están en contra del tema principal. Sin embargo podemos ver otras palabras como *alice* o *vial* que no tienen una razón aparente para pertenecer a esta clase.

```python
from transformers import pipeline
from collections import Counter

# Conjunto de tokens a excluir
exclude_tokens = {',', '.', "'", '-', '"', '[', ']', '0', '(', ')', '[SEP]', '[CLS]'}

def get_important_words(text, model, tokenizer):
    input_ids = tokenizer.encode(text, return_tensors='pt', max_length=512, truncation=True).to(device)
    outputs = model(input_ids, output_attentions=True)
    attentions = outputs[-1]
    last_layer_attentions = attentions[-1][0]
    mean_attentions = last_layer_attentions.mean(dim=0).squeeze()

    # Manejamos múltiples dimensiones si es necesario
    if mean_attentions.dim() > 1:
        mean_attentions = mean_attentions.mean(dim=-1)

    # Convertir los IDs de tokens a tokens
    tokens = tokenizer.convert_ids_to_tokens(input_ids[0])

    # Crear pares de tokens y sus valores de atención, excluyendo los tokens no deseados
    token_attention_pairs = [
        (token, attention.item()) for token, attention in zip(tokens, mean_attentions)
        if token not in exclude_tokens and not token.startswith('##')
    ]

    # Ordenar por atención, de mayor a menor
    sorted_tokens = sorted(token_attention_pairs, key=lambda x: x[1], reverse=True)
    return sorted_tokens

important_words_by_sentiment = {
    'Negativo': [],
    'Neutral': [],
    'Positivo': []
}

# Iteramos sobre el dataframe y obtener las palabras importantes para cada comentario
for index, row in df.iterrows():
    sentiment = row['Sentiment']
    text = row['TextData']
    important_words = get_important_words(text, model, tokenizer)
    # Solo almacenar las palabras (tokens), no los valores de atención
    important_words_by_sentiment[sentiment].extend([word for word, _ in important_words])

    # Contamos las ocurrencias de cada palabra para cada sentimiento
word_counts = {
    sentiment: Counter(word_list)
    for sentiment, word_list in important_words_by_sentiment.items()
}

# Ordenamos las palabras basándose en la frecuencia y crear la lista final
important_words_summary = {
    sentiment: sorted(word_count.items(), key=lambda item: item[1], reverse=True)
    for sentiment, word in word_counts.items()
}

#Imprimimos la lista de palabras importantes por sentimiento junto al número de apariciones
for sentiment, words in important_words_summary.items():
    print(f"Palabras más importantes para los comentarios de clase {sentiment}:")
    i=0
    for word, count in words:
        i+=1
        print(f"{word}: {count}")
        if i>10:
            break
```

Figura 6.7: Código para obtener las palabras más importantes por Sentimiento

```
Palabras más importantes para los comentarios de clase Negative:
be: 1149
not: 727
person: 607
human: 470
think: 402
sam: 398
like: 385
go: 339
make: 331
would: 326
Palabras más importantes para los comentarios de clase Neutral:
be: 354
not: 222
sam: 208
chat: 158
lex: 150
think: 145
ag: 125
human: 121
got: 109
open: 106
Palabras más importantes para los comentarios de clase Positive:
be: 2290
like: 1536
not: 1296
human: 1082
good: 1078
person: 970
sam: 927
think: 883
would: 858
lex: 783
```

Figura 6.8: Palabras más importantes por sentimiento

```python
def get_exclusive_important_words(important_words_summary):
    # Identificar palabras comunes entre sentimientos
    common_words = set()
    all_sentiments = list(important_words_summary.keys())

    for i, sentiment in enumerate(all_sentiments):
        other_sentiments = all_sentiments[:i] + all_sentiments[i+1:]
        words_current_sentiment = set(word for word, _ in important_words_summary[sentiment])

        for other in other_sentiments:
            words_other_sentiment = set(word for word, _ in important_words_summary[other])
            common_words.update(words_current_sentiment.intersection(words_other_sentiment))

    # Excluir palabras comunes y devolver las 10 palabras exclusivas más importantes por sentimiento
    exclusive_important_words = {}

    for sentiment, words in important_words_summary.items():
        exclusive_words = sorted([(word, count) for word, count in words if word not in common_words], key=lambda x: x[1], re
        exclusive_important_words[sentiment] = exclusive_words

    return exclusive_important_words

# Llamada a la nueva función
exclusive_important_words_summary = get_exclusive_important_words(important_words_summary)

# Imprimir palabras exclusivas por sentimiento
for sentiment, words in exclusive_important_words_summary.items():
    print(f"Palabras más importantes exclusivas de comentarios de la clase {sentiment}:")
    for word, count in words:
        print(f"{word}: {count}")
```

Figura 6.9: Código para obtener las palabras más importantes exclusivas por Sentimiento

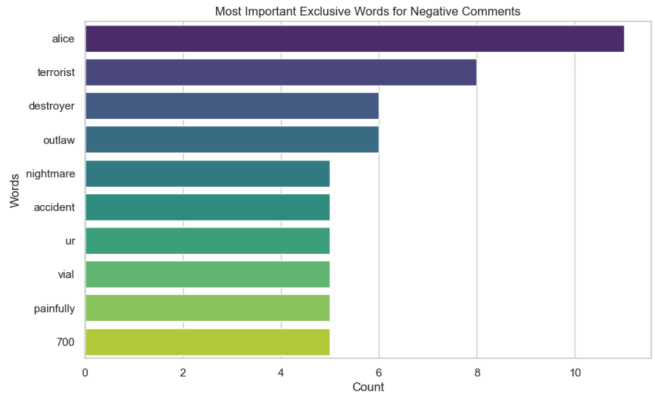

Figura 6.10: Palabras más importantes para la clase negativa

La Figura 6.11 muestra las palabras con mayor *atención* exclusivas de la clase neutral junto a su recuento. Observamos que el *recuento* de las palabras importantes exclusivas de la clase negativa y las de la clase neutral en especial es generalmente bajo por lo que sabemos que son palabras poco comunes dentro de nuestro *dataset*. Esto puede ser un indicio de que la ténica para obtener las palabras más relevantes en la clasificación por sentimientos de nuestros datos no es una solución óptima.

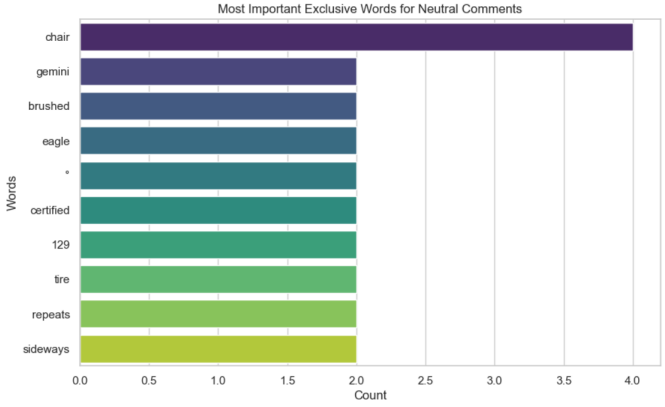

Figura 6.11: Palabras más importantes para la clase neutral

Para el caso de la clase positiva mostrado en la Figura 6.12 observamos un

recuento para las palabras más significante que para las clases anteriores. Este comportamiento es entendible dado el sobremuestreo para este clase en nuestro conjunto de datos como se ha visto anteriormente en este trabajo. Por tanto, los resultados para esta clase son más fiables en cuanto la generalización de las palabras utilizadas para expresar estar a favor de OpenAI. Si observamos cada palabra obtenida se puede ver mayor consistencia en la relación entre las palabras obtenidas y la clasificación positiva de los comentarios que las contienen. Se puede entender que la gente simpatiza con OpenAI por la aparición de palabras como *beautiful*, *bless*, *grateful* y *loving* dada las ventajas tecnológicas deducible por las palabras *productivity*, *technique*, *3d*, *optimal*. Cabe a destacar que la palabra con mayor recuento *author* no tiene una relación directa con un sentimiento positivo. Una posible interpretación puede ser que el nicho de autores es uno de los grupos más beneficiados por el descubrimiento de esta tecnología. Por otra parte está el caso de *oppose* que aparentemente expresa un sentimiento de rechazo hacia el tema pero dados los resultados podemos decir que en nuestros datos siempre ha sido utilizada en un contexto en el que el sentimiento general del comentario en el que aparece es positivo. Un ejemplo de caso en el que *oppose* podría ser *"No tiene sentido* ***oponerse*** *a los avances tecnológicos"*

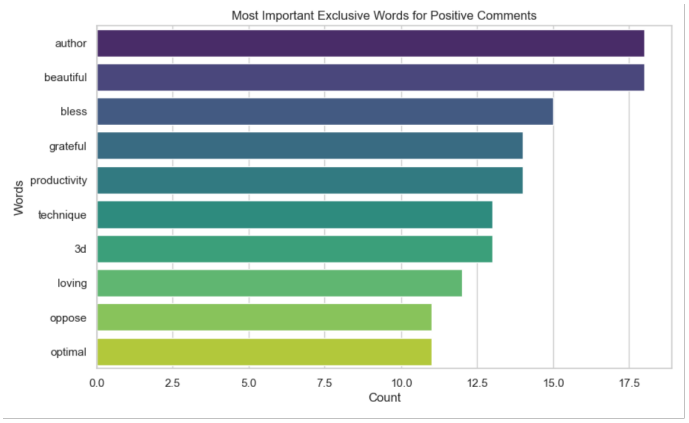

Figura 6.12: Palabras más importantes para la clase positiva

En general, observamos que las palabras importantes de cada clase son generalmente de la índole esperada aunque sabemos que esta es una visión çapadaza que en modelos BERT es muy probable que las palabras varíen su significado según el contexto en el que se utilizan. Palabras que resultan clave para la clasificación en una clase en concreto No estaremos pudiendo visualizarlas en caso de que la palabra aparezca en comentarios con distinta clasificación debido al contexto, lo cual en BERT es altamente probable. Si tomamos la palabra *"bueno"* de ejemplo, puede haber dos comentarios que la utilicen y cuyo significado sea opuesto. Dados los siguientes comentarios de ejemplo:*És un ejemplo bueno'* y *"Parece bueno pero no lo es"*, observamos como el primer ejemplo se consideraría positivo y el segundo negativo al estar negando el sentido de la palabra *"bueno"* por su contexto. La palabra en cuestión tendría un valor de la *atención* elevado en ambos casos pero la clasificación final sería distinta. No obstante, obteniendo las palabras exclusivas para cada sentimiento estamos perdiendo la posibilidad de visualizar casos en los que la atención para un token es elevada pero aparece en comentarios con distinta clasifica-

ción aunque nos aseguramos de librarnos de las palabras contextuales que suelen ser comunes entre clases. No obstante, tuvimos que tomar esta implementación puesto que no hallamos la manera de distinguir las palabras clave que aparecen en más de una clase de las importantes por su contexto, normalmente comunes entre clases, sin tener que analizar cada caso de uso.

Dado lo mencionado, concluimos que la fiabilidad del método para analizar las palabras más importantes con nuestro modelo BERT es notablemente baja puesto que la *atención* y el *recuento* no son variables que ofrezcan unos resultados precisos para el análisis deseado. Por ello, decidimos realizar el análisis de resultados con el modelo para el que obtuvimos el mejor rendimiento después de BERT suponiendo que la complejidad será menor y podremos obtener una solución más fiable y representativa para nuestro propósito.

6.2. Análisis de los resultados del modelo con Regresión Logística

Puesto que el estudio con el modelo BERT no ha sido exitoso debido a su poca transparencia en su método de entrenamiento, utilizaremos el modelo de regresión logística como alternativa para poder hacer un análisis bastante detallado. En nuestro contexto nos interesa ver que palabras han sido más relevantes para cada clase y que polaridad reflejan con dicho nuestro modelo y con la realidad.

Una vez que los datos han sido sometidos al proceso de entrenamiento, es posible obtener los coeficientes asociados a las características del modelo, lo que permite realizar un análisis para, en nuestro caso, determinar qué palabras poseen una mayor relevancia en relación con cada clase del problema de clasificación. Para hacer esto investigamos un poco, y desarrollamos un par de pequeñas herramientas en Python que nos proporcionan información muy útil para analizar más en detalle el comportamiento de nuestro clasificado. La primera herramienta extrae los coeficientes de las características (palabras) del modelo y nos permite visualizar las palabras más importantes, es decir, las palabras con los coeficientes más altos. También podemos visualizar la polaridad y subjetividad real de cada palabra, usando una librería llamada TextBlob orientada para el análisis de texto. Esto nos servirá de referencia real para comparar el sentimiento de las palabras de nuestro modelo con el sentimiento real de las palabras. Finalmente, utilizamos nubes de palabras (*Word Clouds*) para obtener una representación más visual y *user friendly*, que nos permitira ver de una forma más simple que palabras son las más relevantes.Un *word cloud* es una representación visual de palabras en la que la frecuencia de cada palabra se indica mediante su tamaño o color. La segunda herramienta está más orientada al análisis de los errores cometidos por nuestro clasificador.Podremos visualizar instancias en las que nuestro clasificador ha clasificado incorrectamente los comentarios, y así poder ver que palabras pueden ser más conflictivas con nuestro modelo, o como cambia el sentimiento de las palabras segun su contexto.

A continuación se hará un poco mas de hincapié en los métodos utilizados en cada función y su funcionamiento, así como los resultados obtenidos. Posteriormente se realizará un análisis sobre dichos datos.

6.2.1. Análisis de los comentarios con más *likes*

Para realizar este estudio sería interesante ver las tendencias de los comentarios más populares entre la comunidad, de esta forma nos podemos hacer a una del sentimiento general de la gente que interactúa con los vídeos relacionados con este tema y podemos sacar conclusiones que también incluyan las opiniones de gente que no ha comentado. El campo de *likes* añade una nueva dimensión de profundidad a los datos, la popularidad. Para esto, hemos creado unas gráficas para hacernos a una idea del sentimiento mayoritario entre la comunidad. En la Figura 6.13 podemos observar la distribución de sentimiento en los comentarios más populares, con esta gráfica se puede ver claramente que la gran mayoría de los comentarios populares son positivos. Además se puede apreciar que los comentarios negativos y neutrales están muy parejos en cuanto a representación en este conjunto de comentarios.

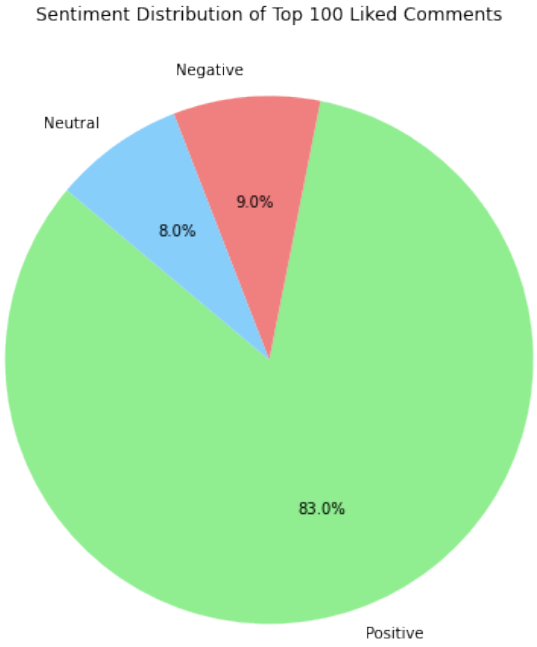

Figura 6.13: Distribución de sentimiento entre los comentarios más populares

También para hacernos a una idea de las palabras más relevantes entre estos comentarios, se ha computado un *WordCloud* para poder apreciar de una manera sencilla las palabras más comunes en este conjunto. Como era de esperar, las palabras que aparecen son prácticamente las mismas que las palabras más populares del conjunto de comentarios positivos, que analizaremos más adelante. El *WordCloud* se puede ver en la Figura 6.14.

En la figura 6.15 podemos observar el código de nuestra función, que realiza los siguientes pasos:

- Definimos una función local para visualizar nuestro WordCloud personalizado que utilizaremos posteriormente. El máximo de palabras representadas esta definido por N, que hemos usado para determinar el numero de palabras de cada clase que queremos analizar, en nuestro caso 25.

Figura 6.14: Palabras más usadas en los comentarios más populares

```python
def generate_wordcloud(ax, text, title):
    wordcloud = WordCloud(width=400, height=200, background_color='white', colormap='Blues',
                          contour_color='steelblue', contour_width=2, max_words=N).generate(text)

    ax.imshow(wordcloud, interpolation='bilinear')
    ax.set_title(title)
    ax.axis('off')

fig, axes = plt.subplots(1, len(pipeline.classes_), figsize=(15, 5))

feature_names = pipeline.named_steps['vectorizer'].get_feature_names_out()
coefficients = pipeline.named_steps['classifier'].coef_

N = 25
for i, sentiment_class in enumerate(pipeline.classes_):
    print(f"\nTop {N} features for class '{sentiment_class}':")
    class_coefficients = coefficients[i]
    top_features_idx = class_coefficients.argsort()[-N:][::-1]
    top_features = [feature_names[idx] for idx in top_features_idx]
    print(top_features)

    print(f"\nAnálisis de Sentimiento para la clase '{sentiment_class}':")
    for word in top_features:
        blob = TextBlob(word)
        print(f"{word}: Polarity={blob.sentiment.polarity}, Subjectivity={blob.sentiment.subjectivity}")

    text_for_wordcloud = ' '.join(top_features)

    generate_wordcloud(axes[i], text_for_wordcloud, f'Word Cloud for Class: {sentiment_class}')
```

Figura 6.15: Código para mostrar las palabras más importantes de cada clase

- Primero se obtienen las características del conjunto de datos vectorizado a traves del pipeline, y a continuación se extraen los coeficientes asociados a cada característica del clasificador.

- Se utiliza un bucle *for* en el que se accede a cada clase del clasificador, y se obtienen las clases mas relevantes para cada clase con class_coefficients.argsort()[-N:][::-1], siendo N el número de palabras de cada clase que queremos almacenar. Una vez hemos obtenido los coeficientes ordenados, almacenamos las caracterñisticas ordenadas por su coeficiente con top_features = [feature_names[idx] for idx in top_features_idx]

- Realizamos otro bucle *for* en el que analizamos con TextBlob el sentimiento de cada palabra, extrayendo su polaridad y subjetividad. Esto nos servirá para discernir si las palabras clasificada por nuestro modelo concuerdan con su sentimiento real (el obtenido con TextBlob).

- Dibujamos las nubes de palbras y las visualizamos con plt.show(). Podemos este código en la Figura 6.15.

6.2.2. Instancias Erróneas

En la Figura 6.16 podemos observar el código que genera las diferentes instancias erróneas y a continuación se detalla como esta elaborado:

```
# Muestra algunas instancias de clasificación incorrecta y analiza características
for idx, (text, true_label, predicted_label) in enumerate(zip(incorrect_predictions_texts[:10], true_labels[:10], predicted_labels[:10])):
    print(f"\nInstancia {idx + 1} (Clase Real: {true_label}, Clase Predicha: {predicted_label}):")
    print(f"Texto: {text}")
```

Figura 6.16: Código para mostrar instancias incorrectas

- Creamos una máscara booleana que indica dónde las predicciones del modelo no coinciden con el sentimiento real de las palabras. Una vez tenemos esta mascara, la usamos para obtener todas las instancias donde la predicción es incorrecta, así como la instancia objetivo real.
- Visualizamos cada instancia con su clase predicha y su clase real. Estas instancias representan los comentarios, aunque las palabras que aparecen están lematizadas. En nuestro caso visualizaremos 10 comentarios clasificados erróneamente, para poder hacer un pequeño análisis de estos errores.

6.2.3. Resultados y análisis

Una vez hemos desarrollado y explicado nuestras herramientas de análisis, podemos ejecutarlas con nuestro clasificador de regresión logística. A continuación se muestran los resultados.

En las Figuras 6.17, 6.18 y 6.19, podemos observar las listas de palabras mas relevantes asi como su polaridad y subjetividad proporcionadas por TextBlob. La polaridad es cuan positivo o negativo es un comentario, siendo negativo -1 y positivo 1. La subjetividad indica cuan objetiva u objetiva es la palabra, siendo 0 totalmente objetiva y 1 totalmente subjetiva.

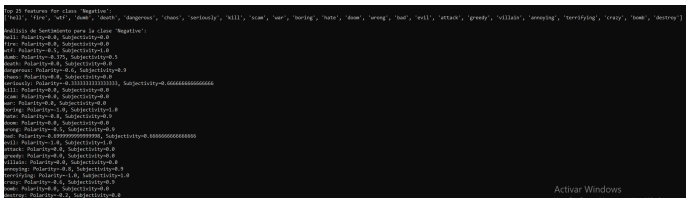

Figura 6.17: Palabras más importantes para la clase negativa

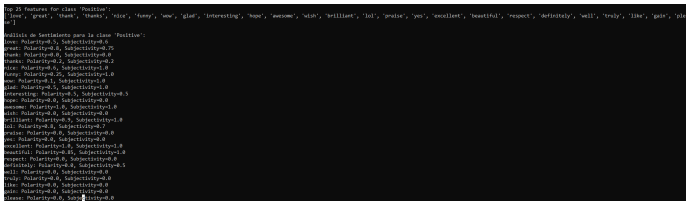

Figura 6.18: Palabras más importantes para la clase positiva

Figura 6.19: Palabras más importantes para la clase neutral

Figura 6.20: Palabras más importantes para las clases negativa, neutral y positiva

Figura 6.21: Instancias erróneas

Podemos observar que en efecto, nuestro modelo clasifica correctamente los sentimientos, pudiendo observar que las palabras de la clase positiva corresponden a sentimientos positivos, como por ejemplo las palabras ***love*** (amor), ***nice*** (bueno), o ***thanks*** (gracias). Lo mismo podemos decir de la clase negativa, con palabras como ***hell*** (infierno), ***chaos*** (caos), o ***scam*** (estafa). La clase neutra también esta acorde, presentando palabras más neutras como ***criticism*** (criticismo), ***career*** (carrera) o ***interface*** (interfaz).

En la Figura 6.20 podemos observar los WordCloud de cada sentimiento, proporcionándonos una visualización de las palabras más sencilla y llamativa.

Los fallos de clasificación nos ayuda a identificar palabras o expresiones con los que el modelo evaluado tiene problemas, como por ejemplo el habla coloquial, jerga de internet o argot. También analizar las predicciones fallidas nos ayuda a detectar posibles sesgos que existan en nuestro modelo que *a priori* no hemos detectado. Toda esta información es de increíble valor a la hora de mejorar los resultados de un modelo en futuras iteraciones. En la Figura 6.21 tenemos las diferentes Instancias de fallos de predicción en texto plano.

A partir de estas instancias hemos extraído las palabras que con más frecuencia se encuentran en textos mal clasificados, con esta información se puede en un futuro intentar mejorar los resultados realizando un estudio más exhaustivo sobre estas palabras. En la Figura 6.22 podemos apreciar estas palabras divididas por sentimiento, y analizando las gráficas nos damos cuenta que hay una serie de palabras que se repiten en las tres gráficas, como ***person*** (persona), ***think*** (pensar), ***be*** (ser/estar).

A la vista de estos resultados, nos pareció útil sacar una única gráfica que muestra

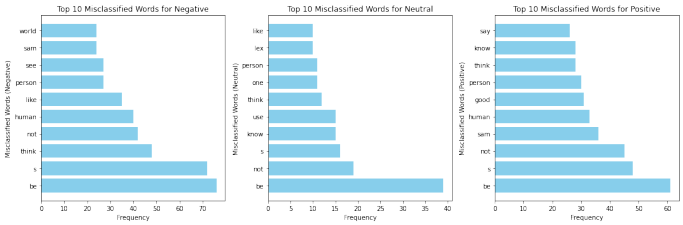

Figura 6.22: Top 10 palabras en textos mal clasificados por sentimiento

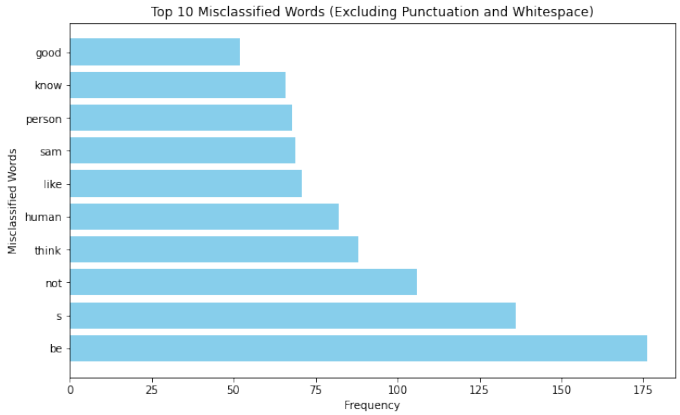

Figura 6.23: Top 10 palabras en textos mal clasificados

las palabras que más se repiten en comentarios mal clasificados que se puede ver en la Figura 6.23. Analizando la gráfica vemos que la mayoría de las palabras son de carácter más neutro, lo cual era de esperar, como ***be*** o ***human***, sin embargo hay otras palabras que son más interesantes. Por ejemplo, la palabra ***sam*** probablemente esté ligada con el nombre del CEO de Open AI, Sam Altman, ya que en el momento del análisis había cierto revuelo con su despido y posterior readmisión en la empresa. Otra palabra que resulta interesante es ***good*** (bueno/bien), esta palabra obviamente tiene un significado positivo, pero es posible que para clasificarla correctamente se necesite de un modelo con una mayor capacidad de analizar el contexto de las palabras.

Capítulo 7

Conclusiones y Trabajo Futuro

En este capitulo final procedemos a dar nuestras conclusiones sobre el trabajo, hablando tanto de todo el proceso de desarrollo del proyecto, como del uso de los distintos clasificadores y los resultados que estos nos han proporcionado. También hablaremos sobre el trabajo futuro en torno a este proyecto.

En primer lugar, todo el proceso previo al entrenamiento de los modelos fue ampliamente interesante, ya que la cantidad de opciones disponibles a la hora de plantear el desarrollo del trabajo es muy extensa, desde el entorno de trabajo a la extracción/obtención de datos. Consideramos que nuestro enfoque del preprocesado le añade un valor extra a todo el proceso, ya que no hemos hemos obtenido un *dataset* ya creado y adaptado para una situación de aprendizaje automático, si no que decidimos elaborar nuestro propio *dataset* desde cero, permitiéndonos comprender cada paso del preprocesado de una forma más profunda, desde la extracción de los propios comentarios a través de una API, pasando por el formateo y limpieza del texto, hasta la tokenización y lematización de nuestro conjunto de datos. Esta herramienta tiene potencial de convertirse en una mucho más compleja, pudiendo realizar tareas más avanzadas, como por ejemplo seleccionar que campos se quieren extraer de los comentarios de YouTube (ubicación, fecha de publicación) o filtrar por idioma que comentarios se quieren extraer, o incluso poder seleccionar un tipo de tokenización y lematización distinta, todo esto acompañado de una pequeña interfaz visual para hacer la herramienta más *user friendly*. Sin duda seguiremos trabajando en esta pequeña herramienta en el futuro.

Dado que el *dataset* en estudio recoge todos los comentarios de los vídeos seleccionados, no tenemos un poder real sobre la distribución de comentarios entre las clases de sentimiento. Además, no es hasta después de entrenar y evaluar los modelos NLP utilizados que se puede apreciar claramente un sobremuestreo de casos positivos en nuestro *dataset*. No obstante, pese a que hay un mejor rendimiento notable para la clase sobrepoblada en todos los modelos utilizados también se obtenemos unos resultados considerablemente altos para las métricas de análisis de la dos clases menos representadas. Pese a que probablemente se podría mejorar el entrenamiento para obtener unos resultados más equilibrados mejorando la proporción de las muestras de las clases, en todos los modelos hemos utilizado técnicas para reducir o mitigar el desnivel de entrenamiento hacia la clase positiva debido a la desproporción favorable

de muestras para esta clase.

En este punto, ya teníamos una idea *grosso modo* de la opinión general de la gente hacia el tema en estudio debido a la proporción de volumen de casos de cada clase. Para comprender bien este comportamiento e intentar fundamentar las posibles causas de los resultados obtenidos continuamos con un análisis profundo que nos permitiera expandir el conocimiento sobre nuestro caso de estudio en concreto.

Dados los resultados de entrenamiento obtenidos para los modelos elegidos fue fácil decretar con qué modelo realizar el análisis profundo dada la notable diferencia de rendimiento entre este modelo y el resto. Sin embargo, resultó ser una opción inviable dada la complejidad de la tarea puesto que la interpretación de modelos BERT para entender la toma de decisiones está en desarrollo en la actualidad y el uso de las herramientas disponibles actualmente que ayudan a implementar la interpretación de modelos personalizados requiere un conocimiento muy elevado en la interpretación de modelos que resultan *cajas negras*, el manejo de datos que hace la herramienta y la interacción real con el modelo a analizar. Dado que no pudimos obtener la atribución de importancia a los tokens analizados por el modelo con *Integrated Gradients* conseguimos hacernos una idea de la clasificación realizada por BERT gracias al estudio de las palabras con mayor puntuación de *atención*. Este paso nos permitió entender la importancia del contexto en BERT dado que las palabras más importantes según la atención para cada clase eran principalmente palabras que recrean la naturaleza del tema a tratar como *human*, *think*, *person*, *open*, *sam* y palabras para comprender el contexto de uso de las palabras más simbólicas de cada comentario como *like*, *not* o *would*. Pese a que nos ayudó a comprender de una manera más visual y realista del funcionamiento de nuestro modelo BERT no obtuvimos una visión de ejemplo de palabras que contribuyesen directamente a la clasificación de un comentario en una clase. Por ello decidimos obtener las palabras exclusivas de cada clase donde si pudimos observar un carácter más claro del sentido de las palabras en relación a su clase otorgada pero conscientes de las limitaciones de esta implementación optamos por utilizar Regresión Logística.

Tras analizar los resultados de la Regresión logística hemos podido comprobar que el sentimiento general de los usuarios de Youtube hacia Open AI es generalmente positivo, haciendo un análisis plano de los resultados, simplemente teniendo en cuenta el global de comentarios, y haciendo un análisis de los comentarios más populares nos han mostrado unos resultados que en su amplia mayoría también son positivos. Es cierto que analizando las instancias erróneas resultantes al aplicar este modelo, hemos detectado que algunas de las palabras conflictivas eran palabras aparentemente de un sentimiento contrario pero debido al contexto en el que se utilizan cambia su significado. Esta falta de contexto es una de las desventajas de este modelo frente al BERT.

A lo largo de la realización de este proyecto hemos adquirido una gran variedad de conocimientos y habilidades relacionadas con el *web-scrapping*, procesado de textos, conceptos probabilísticos y análisis de redes sociales. A pesar de las diferentes problemáticas que han ido surgiendo, hemos sido capaces de reconducir la investigación para conseguir unos resultados esclarecedores que dan luz a la opinión del público general sobre Open AI. Tras el estudio, hemos observado que la opinión general sobre Open AI es abrumadoramente positiva, a pesar de obtener los comen-

tarios de vídeos de diferentes creadores de contenido para reducir el posible sesgo, las opiniones suelen ser similares en cuanto a distribución del sentimiento.

Pese a la satisfacción con el proceso general del proyecto existen ciertos aspectos que podrían ser tomados en cuenta para tratar de mejorar el resultado de los modelos entrenados y la interpretación del más eficiente. Uno de estos aspectos podría ser ampliar el contenido de nuestro *dataset* con variables temporales o geográficas para poder observar la evolución de la opinión pública a lo largo de un período de tiempo y distinguido por zonas geológicas. También se podría realizar una investigación más profunda con más tiempo de estudio sobre técnicas de interpretación de resultados de modelos NLP como BERT, ya que este modelo ofrecía muy buenos resultados y supone una mejora en áreas en las que los otros modelos fallaban más como clasificar correctamente palabras según su contexto.

Introduction

We are in what is probably the era of greatest interpersonal connection in the history of mankind, between the twentieth and twenty-first century great cultural milestones have been achieved, the invention of the radio, the transistor, the microprocessor or the Internet. This last invention has conditioned the present and future of our species in a way that no one could have imagined, for example, no one can imagine a world in which there are no applications like Google Maps to instantly choose the best route in a city we have never been to before. Many people do not even get information through conventional media such as television or newspapers, they get all the information through digital media or social networks. There are social networks entirely dedicated to expressing opinions, such as Twitter, or even Youtube is the habitat of independent media, influencers and other talkers.

The topic chosen for the study is Open AI, we seek to understand how people perceive and react to the company and how their emotions and feelings evolve over time. To achieve this, we will collect a large amount of data using web scraping techniques and classify them according to the sentiment they convey. Subsequently, we will use different classification models to test and validate our analysis with the obtained data.

While this approach focuses specifically on YouTube comments, we believe it will provide valuable insights into sentiment around Open AI. We are confident that the data obtained from these sources will provide a comprehensive understanding of public opinion.

In summary, our study will contribute to the growing body of research on sentiment analysis of social network data and provide insights into public emotions and opinions related to Open AI.

7.1. Motivation

Open AI has gone from being completely unknown by the general public to being one of the most talked about topics in all kinds of discussions and debates, both at an academic level and in more intimate circles. Discussions about this company and its products, ChatGPT being the most relevant of all, range from the future of humanity to theses that propose restructuring learning methods at all stages of the educational cycle. Due to the interest aroused by Open AI in all spheres of society,

it is an ideal topic for a study applying sophisticated natural language processing techniques to try to find and analyze emotions and feelings expressed by users in comments on YouTube videos related to Open AI.

7.2. Objetives

The main purpose of this research is to evaluate the emotions and views that users express on YouTube about the impact of AIs in our society, more specifically the impact of the company OpenAI, developer of the most popular AI today, ChatGPT. Through this detailed analysis, we intend to determine the opinion of society on this topic, as well as to gain deeper insight into the characteristics of the different opinions and to help us understand in depth how these models work and what advantages they have over each other.

Conclusions and Future Work

In this final chapter, we proceed to present our conclusions about the work, discussing both the entire process of project development and the use of various classifiers and the results they have provided. We will also discuss future work related to this project.

Firstly, the entire process leading up to model training was widely interesting, as the range of options available when planning the project development is extensive, from the working environment to data extraction/retrieval. We believe that our preprocessing approach adds extra value to the entire process. Instead of using a pre-existing and adapted dataset for machine learning, we decided to create our own dataset from scratch. This allowed us to understand each step of preprocessing more profoundly, from extracting comments through an API and formatting/cleaning the text to tokenizing and lemmatizing our dataset. This tool has the potential to become much more complex, performing advanced tasks such as selecting specific fields to extract from YouTube comments (location, publication date) or filtering comments by language. It could even allow selecting different tokenization and lemmatization methods, all accompanied by a user-friendly visual interface. We will undoubtedly continue working on this tool in the future.

Since the dataset under study collects all comments from selected videos, we have no real control over the distribution of comments among sentiment classes. Additionally, it is only after training and evaluating the NLP models used that a clear oversampling of positive cases in our dataset becomes apparent. However, despite the notable performance advantage for the overpopulated class in all models used, we also obtain considerably high results for metrics analyzing the two less represented classes. While training could likely be improved to achieve more balanced results by adjusting the sample proportions, we used techniques in all models to reduce or mitigate the training imbalance toward the positive class due to the favorable sample disproportion for this class.

At this point, we already had a general idea of people's opinions on the study topic based on the volume proportion of cases in each class. To further understand this behavior and try to substantiate the possible causes of the results obtained, we continued with a deep analysis to expand knowledge about our specific case study.

Given the training results obtained for the chosen models, it was easy to determine which model to use for the deep analysis, given the significant performance difference between this model and the others. However, it turned out to be an

unfeasible option due to the complexity of the task. Model interpretation for understanding decision-making is still under development for BERT, and currently available tools to implement custom model interpretation require a high level of knowledge in interpreting "black box" models, handling tool-generated data, and actual interaction with the model to analyze. Since we could not obtain token importance attribution analyzed by the model with Integrated Gradients, we gained an understanding of BERT's classification by studying words with the highest attention scores. This step allowed us to grasp the importance of context in BERT, as the most important words for each class according to attention were primarily words that reflect the nature of the topic and words to understand the context of using the most symbolic words in each comment.

Although it helped us understand the functioning of our BERT model in a more visual and realistic way, we did not obtain an example view of words directly contributing to the classification of a comment into a class. Therefore, we decided to extract exclusive words for each class, where we could observe a clearer character of the sense of words in relation to their assigned class, but aware of the limitations of this implementation, we opted to use Logistic Regression.

After analyzing the results of Logistic Regression, we found that the overall sentiment of YouTube users towards OpenAI is generally positive. Simply considering the global comments and analyzing the most popular comments showed results that are mostly positive. Analyzing the resulting misclassified instances revealed that some conflicting words appeared to have an apparently opposite sentiment, but due to the context in which they were used, their meaning changed. This lack of context is one of the disadvantages of this model compared to BERT.

Throughout the project, we acquired a variety of knowledge and skills related to web scraping, text processing, probabilistic concepts, and social network analysis. Despite the various challenges that arose, we were able to redirect the research to achieve enlightening results that shed light on the general public's opinion about OpenAI. After the study, we observed that the overall opinion about OpenAI is overwhelmingly positive. Despite obtaining comments from videos of different content creators to reduce potential bias, opinions tend to be similar regarding sentiment distribution.

Despite satisfaction with the overall project process, there are certain aspects that could be considered to try to improve the results of trained models and the interpretation of the most efficient one. One of these aspects could be expanding the content of our dataset with temporal or geographical variables to observe the evolution of public opinion over a period of time and distinguished by geographical areas. A more in-depth investigation could also be conducted with more study time on techniques for interpreting results of NLP models like BERT, as this model offered very good results and represents an improvement in areas where other models, such as correctly classifying words based on their context, were more challenging.

7.

Contribuciones Personales

Antes de comenzar con el esquema de las contribuciones individuales explicadas más en detalle cabe destacar que gran parte del trabajo se ha hecho de manera conjunta o cooperativa siendo todos los integrantes del proyecto conscientes del proceso llevado a cabo en cada apartado del mismo. No obstante, el desarrollo de tareas específicas del trabajo se decidió distribuir de manera equitativa conservando la comunicación del equipo para mantener una visión global del proyecto.

Jorge Villacorta

- Investigación previa al comienzo del TFG. Búsqueda de herramientas y preparación del entorno de trabajo en local (Python 3.9 + CMD Windows)
- Investigación sobre la extracción de texto y herramientas y técnicas de preprocesado (Octoparse y software similar, API de Youtube, NLTK).
- Encargado del planteamiento y desarrollo de la herramienta de preprocesado, tanto el script principal como las etapas/pasos internos de este (Python).
- Desarrollo, entrenamiento y análisis de resultados de los modelos Random Forest y Regresión Logística Multiclase (Python)
- Realización de la memoria (Overleaf)
- Realización del análisis en detalle de el modelo de regresión logística así como las herramientas de análisis de datos utilizadas.

Alejandro Tabernero

- Investigación inicial, intentar extraer datos de Twitter, investigar posibles bibliotecas para realizar el análisis(Textblob, Stanza) y realizar pruebas con dichas bibliotecas.
- Contribuidor en el perfeccionamiento de la herramienta de preprocesado.

- Estudio de conceptos probabilísticos y posibles modelos a aplicables al proyecto.
- Desarrollo, entrenamiento y análisis de resultados de los modelos K-Nearest Neighbours y Multinomial Naive Bayes.
- Realización del análisis en detalle de el modelo de regresión logística así como las herramientas de análisis de datos utilizadas.
- Realización de la memoria

Alonso Mata

- Investigación de modelos NLP para el análisis de sentimientos.
- Contribuidor en el perfeccionamiento de la herramienta de preprocesado.
- Estudio y comprensión de conceptos clave para la implementación de los modelos (Vectores de soporte, Arquitectura Transformes, Tensores, Función Softmax, etc).
- Desarrollo, entrenamiento y análisis de resultados de los modelos SVM y BERT.
- Análisis profundo de resultados del modelo BERT
- Realización de la memoria

Personal Contributions

Before detailing the personal contributions it should be noted that the majority of the project has been done cooperatively, with all members of the proyect being aware of the processes carried out in each section of the project. However, the development of some task was distributed equitably among the members of the group while maintaining communication in order to get a global vision of the project.

Jorge Villacorta

- Initial research, Searching for tools and preparation of the local work environment (Python 3.9 + CMD Windows).
- Research on text extraction and preprocessing tools and techniques (Octoparse and similar software, API de Youtube, NLTK).
- Responsible for the approach and development of the preprocessing tool, both the main script and the internal stages/steps of the tool (Python).
- Development, training and result analysis of Random Forest and Multiclass Logistic Regression models (Python).
- Reporting tasks (Overleaf).
- Detailed analysis of the logistic regression model as well as the data analysis tools used.

Alejandro Tabernero

- Initial research, Twitter data extraction research, research different possible libraries to use (Textblob, Stanza) y testing of said libraries.
- Contributor in the improvement of the preprocessing tool.
- Study of probabilistic concepts and possible models to be applied to the project.

- Development, training and analysis of model results K-Nearest Neighbours y Multinomial Naive Bayes.
- Detailed analysis of the logistic regression model as well as the data analysis tools used.
- Reporting tasks.

Alonso Mata

- Investigation of NLP models for sentiment analysis.
- Contributor in the improvement of the preprocessing tool.
- Study and understanding of key concepts for the implementation of the models (Support Vectors, Transform Architecture, Tensors, Softmax Function, etc).
- Development, training and results analysis of SVM and BERT models.
- In-depth analysis of BERT model results.
- Reporting tasks (Overleaf).

Bibliografía

[1] scikit learn, Support vector machines, https://scikit-learn.org/stable/modules/svm.html, [Online; accessed 25-November-2023] (2023).

[2] B. Muller, Bert 101 state of the art nlp model explained, https://huggingface.co/blog/bert-101, [Online; accessed 8-December-2023] (2022).

[3] D. of Electrical, D. U. D. Computer Engineering, Syntax-infused transformer and bert models for machine translation and natural language understanding, https://arxiv.org/pdf/1911.06156v1.pdf, [Online; accessed 27-December-2023] (2019).

[4] P. S. Foundation, Interfaces misceláneas del sistema operativo, https://docs.python.org/es/3/library/os.html, [Online; accessed 15-November-2023] (2023).

[5] P. S. Foundation, Gestión de subprocesos, https://docs.python.org/es/3/library/subprocess.html, [Online; accessed 15-November-2023] (2023).

[6] K. Reitz, Requests: Http for humans, https://requests.readthedocs.io, [Online; accessed 15-November-2023] (2023).

[7] P. S. Foundation, Hypertext markup language support, https://docs.python.org/3/library/html.html, [Online; accessed 15-November-2023] (2023).

[8] E. R. Schmidt, Expresiones regulares, https://rico-schmidt.name/pymotw-3/re/index.html, [Online; accessed 3-January-2023] (2019).

[9] P. S. Foundation, Operaciones con expresiones regulares, https://docs.python.org/es/3/library/re.html, [Online; accessed 3-January-2023] (2023).

[10] M. Danilk, langdetect 1.0.9, https://pypi.org/project/langdetect/, [Online; 15-November-2023] (2021).

[11] N. Project, Natural language toolkit, https://www.nltk.org/, [Online; 15-November-2023] (2023).

[12] Wikipedia, Nltk, https://es.wikipedia.org/wiki/NLTK, [Online; 15-November-2023] (2023).

[13] S. N. Group, Stanza – a python nlp package for many human languages, https://stanfordnlp.github.io/stanza/, [Online; 15-November-2023] (2020).

[14] C. U. Press, Stemming and lemmatization, https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html, [Online; accessed 7-December-2023] (2008).

[15] MathWorks, Stemming, https://la.mathworks.com/discovery/stemming.html, [Online; accessed 4-December-2023] (1994).

[16] S. Srinidhi, Lemmatization in natural language processing, https://builtin.com/machine-learning/lemmatization, [Online; accessed 6-December-2023] (2023).

[17] A. S. Gillis, lemmatization, https://www.techtarget.com/searchenterpriseai/definition/lemmatization, [Online; accessed 6-December-2023] (2023).

[18] P. Jain, Basics of countvectorizer, https://towardsdatascience.com/basics-of-countvectorizer-e26677900f9c, [Online; accessed 19-December-2023] (2021).

[19] scikit learn, sklearn.featureextraction.text.tfidfvectorizer, https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html, [Online; accessed 19-December-2023] (2021).

[20] D. L. Yse, K-nearest neighbor (knn) explained, https://www.pinecone.io/learn/k-nearest-neighbor/, [Online; accessed 8-December-2023] (2023).

[21] T. Srivastava, A complete guide to k-nearest neighbors, https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/, [Online; accessed 8-December-2023] (2023).

[22] ibm, K-nearest neighbors algorithm, https://www.ibm.com/topics/knn, [Online; accessed 8-December-2023] (2023).

[23] E. Gomede, Understanding multinomial naive bayes classifier, https://medium.com/@evertongomede/understanding-multinomial-naive-bayes-classifier-fdbd41b405bf, [Online; accessed 8-December-2023] (2023).

[24] scikit learn, Multinomial naive bayes, https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes, [Online; accessed 8-December-2023] (2023).

[25] Wikipedia, Random forest, https://en.wikipedia.org/wiki/Random_forest, [Online; accessed 1-December-2023] (2023).

[26] scikit learn, sklearn.ensemble.randomforestclassifier, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, [Online; accessed 1-December-2023] (2023).

[27] Wikipedia, Validación cruzada, https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada, [Online; accessed 3-January-2023] (2023).

[28] scikit learn, Cross-validation: evaluating estimator performance, https://scikit-learn.org/stable/modules/cross_validation.html, [Online; accessed 3-January-2023] (2023).

[29] scikit learn, sklearn.pipeline.pipeline, https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html, [Online; accessed 19-December-2023] (2021).

[30] scikit learn, sklearn.model-selection.train-test-split, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html, [Online; accessed 21-December-2023] (2023).

[31] scikit learn, sklearn.metrics.classificationreport, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classificationreport.html#sklearn.metrics.classificationreport, [Online; accessed 15-December-2023] (2015).

[32] scikit learn, sklearn.linear.model.logisticregression, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, [Online; accessed 21-December-2023] (2023).

[33] Wikipedia, Función softmax, https://es.wikipedia.org/wiki/Funci%C3%B3n_SoftMax, [Online; accessed 3-January-2024] (2022).

[34] E. J.Carmona, Tutorial sobre máquinas de vectores soporte (svm), http://www.ia.uned.es/~ejcarmona/publicaciones/%5B2013-Carmona%5D%20SVM.pdf, [Online; accessed 27-November-2023] (2014).

[35] scikit learn, sklearn.svm.svc, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC, [Online; accessed 10-December-2023] (2023).

[36] V. Reddy, Sentiment analysis using svm, https://medium.com/scrapehero/sentiment-analysis-using-svm-338d418e3ff1, [Online; accessed 10-December-2023] (2018).

[37] scikit learn, sklearn.multiclass.onevsrestclassifier, https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html, [Online; accessed 10-December-2023] (2024).

[38] HuggingFace, Pytorch-transformers, https://pytorch.org/hub/huggingface_pytorch-transformers/, [Online; accessed 15-December-2023] (2022).

[39] TensorFlow, Clasificar texto con bert, https://www.tensorflow.org/text/tutorials/classify_text_with_bert?hl=es-419, [Online; accessed 15-December-2023] (2022).

[40] NvidiaDeveloper, Cuda toolkit 11.1.0, https://developer.nvidia.com/cuda-11.1.0-download-archive?, [Online; accessed 19-December-2023] (N/A).

[41] NvidiaDeveloper, cudnn archive, https://developer.nvidia.com/rdp/cudnn-archive, [Online; accessed 19-December-2023] (N/A).

[42] NvidiaDocsHub, Installation guide, https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html, [Online; accessed 19-December-2023] (2023).

[43] scikit learn, Metrics and scoring: quantifying the quality of predictions, https://scikit-learn.org/stable/modules/model_evaluation.html, [Online; accessed 15-December-2023] (2015).

[44] scikit learn, sklearn.metrics.accuracy-score, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html, [Online; accessed 21-December-2023] (2023).

[45] scikit learn, sklearn.metrics.precision-recall-fscore-support, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support, [Online; accessed 15-December-2023] (2015).

[46] S. Salih, Curse of dimensionality: An intuitive exploration, https://towardsdatascience.com/curse-of-dimensionality-an-intuitive-exploration-1fbf155e1411, [Online; accessed 1-January-2024] (2023).

[47] G. E. Nikolaos Kouiroukidis, The effects of dimensionality curse in high dimensional knn search, https://ieeexplore.ieee.org/document/6065061, [Online; accessed 1-January-2024] (2011).

[48] S. S. Nalla, Explainable ai: Interpreting bert model, https://medium.com/@sn7d4/explainable-ai-interpreting-bert-model-534094e0e015, [Online; accessed 27-December-2023] (2022).

[49] M. Ancona, E. Ceolini, C. Öztireli, M. Gross, Towards better understanding of gradient-based attribution methods for deep neural networks, https://towardsdatascience.com/using-a-dataloader-in-hugging-face-52388f552259, [Online; accessed 27-December-2023] (2018).

[50] Captum, Integrated gradients, https://captum.ai/docs/extension/integrated_gradients, [Online; accessed 27-December-2023] (2024).

[51] N. Katz, Using a dataloader in hugging face, https://towardsdatascience.com/using-a-dataloader-in-hugging-face-52388f552259, [Online; accessed 27-December-2023] (2021).

[52] Seaborn, seaborn, statistical data visualization, https://seaborn.pydata.org/index.html, [Online; accessed 27-December-2023] (2023).