MÁSTER EN DATA SCIENCE & BUSINESS ANALYTICS ONLINE

Análisis de Sentimientos para la Gestión y Mejora de la calidad de las ventas de productos de alimentación ofrecidos en Amazon

TFM elaborado por: Saúl Rozada Raneros

Tutor de TFM: Juan Manuel Moreno Lamparero

Valladolid a enero de 2025

Resumen

El presente proyecto trata de aplicar técnicas de Procesamiento de Lenguaje Natural (NLP) para analizar las reseñas de productos de alimentación en Amazon. Utilizando el conjunto de datos "Amazon Fine Food Reviews", compuesto por más de 568,000 registros, el objetivo es clasificar productos en base a opiniones de los usuarios obteniendo el sentimiento de la reseña para conocer si es positiva, negativa o neutra y con ello, proponer estrategias de mejora de las recomendaciones y ventas en la plataforma. Este estudio busca no solo identificar productos mejor valorados, sino también optimizar la experiencia del cliente y la calidad de los productos ofrecidos, proporcionando retroalimentación valiosa tanto para Amazon como para los vendedores. Para ello, se hará uso de la metodología CRISP-DM, en la que se incluye preprocesamiento de texto (limpieza, tokenización y lematización), transformación de características mediante TF-IDF así como evaluación con modelos de Machine Learning simples y elaborados, mediante técnicas de aprendizaje supervisado basadas en las etiquetas que proporcionan las puntuaciones de las reseñas de los productos.

Abstract

The present project tries to apply Natural Language Processing (NLP) techniques to analyze food product reviews on Amazon. Using the “Amazon Fine Food Reviews” dataset, composed of more than 568,000 records, the objective is to classify products based on user opinions by obtaining the sentiment of the review to know if it is positive, negative or neutral and thus, propose strategies to improve recommendations and sales on the platform. This study seeks not only to identify top-rated products, but also to optimize the customer experience and the quality of the products offered, providing valuable feedback for both Amazon and sellers. To do so, it will make use of the CRISP-DM methodology, which includes text preprocessing (cleaning, tokenization and lemmatization), feature transformation using TF-IDF as well as evaluation with simple and elaborated Machine Learning models, using supervised learning techniques based on the labels provided by product review scores.

Índice

Introducción y antecedentes

El análisis de sentimientos es una técnica de procesamiento de lenguaje natural o NLP, qué permite evaluar las emociones expresadas en los textos escritos. Esto es realmente útil en el ámbito de aplicaciones digitales como el comercio electrónico, redes sociales… dónde la opinión de las personas es fundamental para evaluar ya sea la calidad de los productos que se venden en una plataforma y tomar decisiones de qué productos ofrecer de manera más eficiente, o bien en el plano de las redes sociales encontrar el patrón de comportamiento de determinadas personas o de interactuación de estas personas con otros.

En el caso del presente proyecto se aborda la implementación de un modelo de análisis de sentimientos sobre un conjunto de datos masivo de reseñas de productos alimenticios obtenidos de Amazon. Mediante técnicas de procesamiento de lenguaje natural y aprendizaje automático se buscará categorizar las reseñas en positivas, negativas y neutrales, ayudando al desarrollo de estrategias que mejoren la calidad de los productos vendidos en la plataforma y a su vez optimice la experiencia del usuario en la misma.

Amazon es una de las mayores empresas a nivel global, fundada en el año 1994 dónde comenzó siendo una librería en línea, pero que ha ido diversificándose hasta llegar a vender toda clase de productos online siendo una de las empresas más importantes a nivel mundial en el sector de la venta online. Eso lo consigue gracias a un gran número de técnicas de mejora de negocio que han ido aplicando a lo largo de los años, sobre todo en eficiencia en venta, en análisis de datos…

Las bases de su modelo de negocio en la actualidad, radican en los siguientes puntos según apunta Pereira D. (2024). El primero es el comercio electrónico (E-Commerce), ya que, como se ha comentado con anterioridad, Amazon ofrece una amplia gama de productos desde libros hasta electrónica, ropa… a través de su plataforma online. Amazon también ofrece servicios en la nube con Amazon Web Services (AWS). Con esto proporciona servicios de computación en la nube a empresas y organizaciones a nivel mundial. Está división de su negocio ha experimentado un grandísimo crecimiento contribuyendo de una manera muy importante a los ingresos y beneficios de la compañía.

En esta diversificación de servicios como identifica Beucher M. (2024), aparece Amazon Prime un servicio de envíos rápidos bajo suscripción, Kindle para la distribución de libros electrónicos, y Amazon Fresh para la entrega de productos alimenticios. A esto hay que añadir que siendo cliente de Amazon Prime, el cliente puede disfrutar de la plataforma de televisión de Amazon con series y películas exclusivas. Se centra mucho en el enfoque en el cliente puesto que trata de ofrecer la mejor experiencia de usuario con precios competitivos, entregas rápidas y un gran servicio al cliente, que sea lo más eficiente posible. Esto hace que se genere una gran lealtad del consumidor a sus servicios. Esta estrategia ha hecho que le lleve a ser una de las mejores empresas E-commerce mundiales.

Objetivos del proyecto

En este proyecto se va a tratar de estudiar es cómo se podría realizar un proyecto de análisis de los sentimientos (NLP) aplicado sobre reseñas de productos de Amazon. Para ello se hará uso de todo lo que se ha visto durante el máster y sobre todo realizar un entendimiento conjunto de todos los conceptos que se han ido obteniendo con del desarrollo de este máster.

- Obtener una visión general de todos los conceptos que se han ido estudiando durante todo el máster

- Resolución de dudas y problemas que vayan apareciendo a lo largo del proyecto con un mínimo de mirada crítica y justificación sobre todos los conceptos que se han ido consiguiendo a lo largo del estudio de este máster además de documentación externa
- Obtener un conocimiento general de lo que conlleva la parte analítica de un proyecto de Data Science, aplicando técnicas NLP.
- Comprender y aplicar métodos de preprocesamiento de texto para la extracción y limpieza de datos
- Adquirir destreza en la utilización de algoritmos de clasificación y técnicas de Machine Learning para el análisis de sentimientos.
- Desarrollar habilidades para evaluar la eficacia de un modelo mediante el uso de métricas de rendimiento y técnicas de validación cruzada.
- Desarrollar un enfoque crítico en el diseño y análisis de modelos, siendo capaz de identificar puntos de mejora y realizar ajustes a los algoritmos.
- Aprender a presentar y comunicar de forma efectiva los resultados del análisis, así como las recomendaciones para la mejora de productos.

Material y métodos

El método a seguir sería el siguiente:

a) Comprensión del negocio y de los datos implicados en el análisis.
b) Preprocesamiento de texto mediante técnicas como eliminar ruido, stopwords, lemanización, etc… de manera que el texto quede preparado para realizar el análisis.
c) Representación del texto mediante técnicas TF-IDF o Word Embeddings que permitan convertir los textos en características numéricas que puedan ser procesadas por los modelos
d) Se probarán diferentes modelos de Machine Learning como pueden ser Logistic Regression, Random Forest, la utilización de la librería HuggingFace… Con el objetivo de observar el comportamiento de las distintas técnicas con este tipo de datos. El enfoque se basará en un modelo supervisado, utilizando como etiquetas las valoraciones de las reseñas.
e) Evaluar el rendimiento del modelo con métricas como Precision, Recall, F1-Score y AUC. Se realizarán pruebas con nuevos datos para validar el desempeño y robustez de las nuevas reseñas.

En cuanto al material, se va a utilizar en primer lugar un dataset de Kaggle qué es una lista de reviews de productos de alimentación de Amazon que han sido conseguidos a lo largo de 10 años de distintas personas y sobre distintos productos. Con ello lo que va a permitir, es partir de un conjunto de datos lo suficientemente grande como para poder desarrollar el proyecto y todos los conceptos que se pretenden emplear sobre este conjunto de datos sin ningún tipo de problemas.

Se hará uso de Google Colab en su versión gratuita como entorno de trabajo, esto es importante ya que ofrece un entorno en el que no hay que preocuparse por las dependencias que se deben de instalar en un entorno local y abstrae de todo tipo de tecnología que se deba tener instalada en nuestro equipo local, con lo que ahorra un una gran cantidad de tiempo a la hora de verificar las distintas compatibilidades o incompatibilidades que deben de tener los paquetes, las versiones de distintos paquetes en las versiones de Python utilizadas, además de no depender de la posible falta de potencia de nuestro equipo local para este tipo de tareas. Con ello, el desarrollo del proyecto se realiza de una manera mucho más cómoda y rápida. Este también es

un punto muy interesante, ya que en un entorno empresarial real muchas veces se debe enfrentar multitud de problemas y limitaciones con el entorno de trabajo, debido a las especificaciones que tiene una determinada empresa o institución. Por tanto, es una muy buena oportunidad, para entender y comprender los problemas que se pueden presentar en entornos con recursos limitados.

También se hará uso de los distintos materiales aportados durante el máster en cuanto a posibles dudas o consideraciones que se puedan ir llevando a cabo en el proyecto. Se revisarán distintos conceptos vistos a lo largo de las unidades didácticas del máster para poder llegar a al análisis que se pretende realizar con el desarrollo de este proyecto. Además, es importante también una correcta utilización de documentos o referencias bibliográficas, o incluso que se puedan encontrar en la web, a fin de realizar también tareas de investigación y documentación en el desarrollo de un proyecto real.

Finalmente, también se hará uso de una librería denominada Transformers de la empresa HuggingFace, que es referente en aplicaciones de inteligencia artificial y sobre todo en el ámbito del procesamiento de lenguaje natural. Esto ayudará a conformar un análisis desde el punto de vista de transferencia de aprendizaje con redes neuronales.

Resultados

Comprensión del negocio

Uno de los pilares de Amazon para ser el número 1 en E-commerce a nivel mundial, se basa en ser muy eficiente y en una estrategia de precios de bajo coste, es decir, la mejor o mejores ofertas de productos en el mercado siendo muy competitivo. Todo ello sin despreciar la exquisita relación con los clientes de la plataforma.

Por tanto, este proyecto nace como una idea que trata de ayudar a Amazon a ser más eficiente a la hora de seleccionar los productos alimenticios que se van a promocionar o se van a permitir vender en su plataforma. Con ello, se conseguirá promocionar y mejorar la venta de productos que estén muy bien valorados por los usuarios y penalizar en posiciones menos visibles de la web, a aquellos productos que no están muy bien valorados por los usuarios. A su vez Amazon permitiría hacer reportes a los clientes de sus productos para que pudieran obtener feedback y así poder mejorar dichos productos de cara a futuro. Esto permite obtener una estrategia Win-Win dónde el creador del producto puede obtener una realimentación que es muy útil de cara a la mejora de su producto, pero a su vez también permite aumentar su competencia frente a otros creadores de productos similares, lo que permite aumentar la calidad a largo plazo de los productos que se pueden vender en Amazon.

Por otra parte, Amazon consigue mejorar las recomendaciones de los productos, posicionando los mejores productos en los primeros resultados de búsqueda de productos en su web, y aumentar la calidad de sus ventas ya que va a dedicar más esfuerzo a promocionar esos productos que están mejor valorados. Así el usuario que entra a Amazon buscando un determinado producto en concreto y/o de una categoría en particular, consigue la mejor opción posible limitando el tiempo de búsqueda de la mejor opción de compra o comparación de productos similares.

Para poder realizar este estudio se va a aplicar procesamiento de lenguaje natural (NLP), para poder evaluar los distintos comentarios que realizan los usuarios acerca de sus productos esto

es una técnica muy útil a la hora de clasificar las distintas opiniones que tienen los usuarios en función de las compras que han realizado en Amazon.

Posteriormente, se dará una descripción más detallada del conjunto de datos que se están utilizando para el presente estudio, pero atendiendo a algunos de los datos que aparecen en dicho conjunto, se pueden calcular gráficas que muestran el valor que tienen estos datos. En dicho conjunto, aparecen datos referentes a la utilidad de las reseñas escritas por otros clientes en la plataforma sobre un producto en concreto y el número total de usuarios que calificaron esa reseña ya sea como útil o no útil.

Con ello, se puede tomar la clasificación de las reseñas que han sido útiles para los usuarios, y aquellas reseñas que no han sido útiles para los usuarios, también se incluye en el gráfico aquellas personas que no han respondido si se aprecia que más del 94% ha encontrado en las reseñas que son útiles o no son útiles es decir que los usuarios que han visto dicho producto, se han fijado en las reseñas a la hora de valorar la compra, o no de dicho producto de comida. Por tanto, se dispone de un conjunto de datos muy potente a la hora de evaluar si un producto es bueno o no es bueno y por tanto poder obtener información acerca de los productos que se venden en Amazon. Se verá más adelante con el progreso de este proyecto, qué datos son los que se van a ir tomando para la realización del estudio y cuáles son los de gran utilidad para poder realizar el análisis de las reseñas.

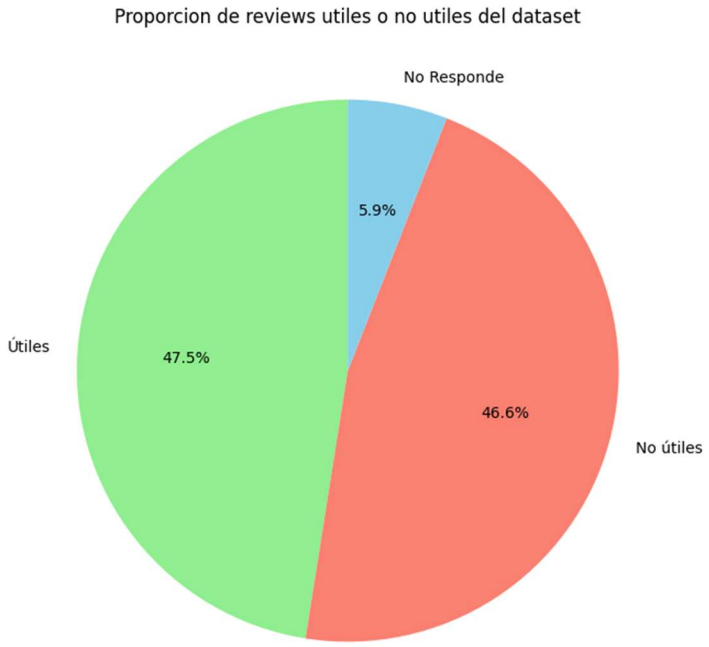


Figura 1. Porcentaje de usuarios que encuentran útiles o no las reseñas

Si se atiende a un cálculo sobre la distribución de las reseñas como se ve en la Figura 1, se aprecia que está bastante equiparado entre aquellas que son buenas, que son mayores a 3, malas, que son menores a 3, y hay algunas que son neutras que representan en torno a la mitad de estas dos primeras.

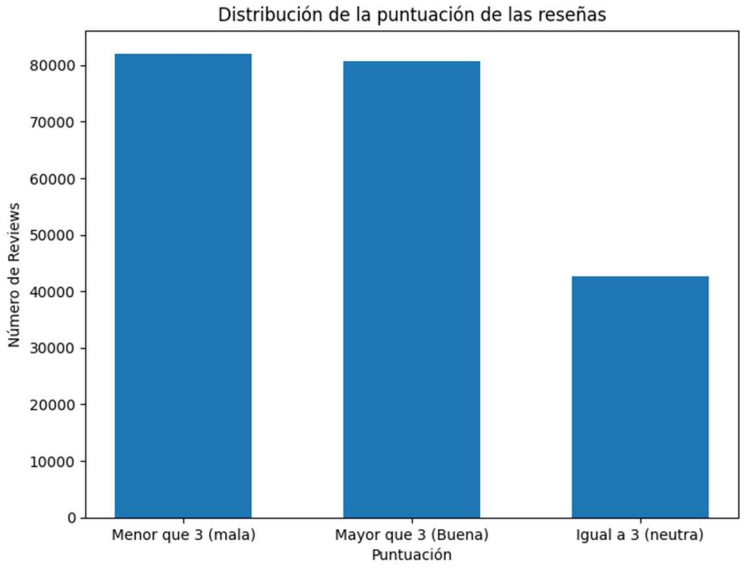


Figura 2. Distribución total de la puntuación de las reseñas del conjunto de datos

Es decir, se abstrae de la Figura 2, qué hay muchas personas que realizan una puntuación de las distintas reseñas y sobre todo añade en un comentario en cada una de las reseñas por tanto sí que se puede obtener bastante información acerca de cómo es el producto que se está vendiendo de si es bueno o malo. Todos estos análisis se profundizarán en las siguientes secciones, pero en este primer análisis de negocio se puede ver que los datos de los que se dispone, son bastantes útiles para obtener realimentación de los distintos productos de alimentación que se están vendiendo en Amazon.

Adquisición de las fuentes de datos

En cuanto al conjunto de datos que se va a utilizar para el desarrollo del proyecto, se dispone de un conjunto de datos de Kaggle sobre las reseñas de productos de comida que se venden en Amazon: *Amazon fine food reviews* obtenido de Kaggle. (n.d.), que contiene suficiente información para poder desarrollar el proyecto con relativa solvencia.

Se trata de un dataset que viene en formato *CSV* de 300 Megabytes de tamaño, y con unas dimensiones de 568.454 filas y 10 columnas, lo que da una gran cantidad de registros para poder realizar el proyecto. Más adelante se realizará una descripción más detallada de los campos de los que está compuesto, pero de lo que sí que se puede describir ya es que se trata de un conjunto de datos estructurado, ya que viene delimitado por la estructura dada por el propio documento CSV.

De este conjunto de datos, no va a ser utilizado todo como se va a explicar a continuación ya que, depende de las características de las que se disponen en cuanto a recursos que se están usando, como Colab en su versión gratuita. Se dispone de más de 560.000 filas, lo que implica una grandísima cantidad de datos qué para la versión de Google con la app gratuita puede ser excesiva a la hora de realizar entrenamientos u operaciones de gran coste computacional, por lo que se debe pensar que ciertas técnicas que se van a utilizar, y que serán descritas más adelante, como por ejemplo pueden ser árboles de decisión para poder estimar la variable objetivo, y que lleva un tiempo de computación bajo aunque sea muy costoso computacionalmente para un gran conjunto de datos.

Sin embargo, se puede dar el caso al usar redes neuronales, y en ese caso dependiendo del número de hiperparámetros puede darse que los entrenamientos con esta gran cantidad de filas, se puedan demorar bastantes horas, y esto implica que en la versión gratuita de Colab que tiene en torno a unas pocas horas aproximadamente de sesión, se pueda cortar la sesión durante el entrenamiento. Por tanto, para evitar esta disyuntiva lo que se ha barajado han sido varios escenarios:

El primero de ellos se trataría de tomar el producto que se desea evaluar de todo ese conjunto de datos y realizar todo el estudio en torno a ese único producto. El problema es que al obtener las filas que corresponden a uno de esos productos, el número de muestras que se obtiene es muy pequeño, en el mejor de los casos se pueden obtener en el entorno de las 900 filas y esto es un conjunto de datos muy pequeño, por lo que no sería demasiado idóneo, ya que, para el entrenamiento de datos mediante en NLP, sería más interesante disponer de una mayor cantidad de datos, ya que, las características del lenguaje permitirían obtener mejores resultados.

Otro escenario que se ha descrito anteriormente, sería tomar todas las filas de las que se dispone, pero como se ha indicado para la versión de Colab gratuita es bastante tedioso, y puede dar lugar a errores y problemas. Por tanto, esta opción queda también descartada, salvo en un escenario en el que se dispone de un entorno en el que se pueda entrenar las 568.000 filas de datos sería lo idóneo tomar toda la gran cantidad de datos

Finalmente se puede plantear un escenario intermedio, que sería un escenario en el que se tomasen una serie de productos con el fin de aumentar el número de muestras con las que se puede llegar a realizar el entrenamiento, pero sin llegar a un número de registros tal que pueda cancelar la sesión en Colab gratuito. Esto asegura una estrategia suficientemente buena para realizar todo el análisis, y se usarán varios de estos 250 productos que se van a seleccionar, para realizar la evaluación de los productos, es decir, el objetivo del proyecto es evaluar cómo de bueno es un conjunto de productos y clasificarlos en si son productos buenos o son productos malos. Por tanto, para realizar el entrenamiento se usarán de esta cantidad de productos una porción y se evaluará únicamente con otro subconjunto de ellos, que serán los productos de los cuales se deseen obtener esa información de si es bueno o es malo. Todo esto se explicará de nuevo más adelante con más detalle para que quede más claro.

Por tanto, se ha optado por esta última estrategia, es decir, tomar en torno a unos 250 productos, que representan el 15% del dataset original, y configurar un conjunto de datos más pequeño sobre los que se puede realizar el estudio que se va a desarrollar. Se puede tomar este proyecto, como todo el conjunto de procesos a realizar hasta una etapa de modelado, ya que es recomendable en muchos casos primero tomar una serie de datos al azar para realizar el modelo y luego generalizar al resto de datos tal y como afirma Conroy (2023). Una vez que ya se ha generado el modelo y ya se han validado los estudios… Se procede a generalizar al conjunto de todos los datos. Pero esta ya sería una fase muy posterior al desarrollo del ámbito de este proyecto.

Así pues, para la obtención de este conjunto de datos lo que se realiza, es sobre el dataframe utilizar la función *head* de forma que nos devuelva los 250 productos que tienen más opiniones. De esta forma se va a conseguir un subconjunto de datos partiendo del inicial mucho menor y mucho más manejable, pero que asegurará los suficientes datos como para poder realizar unos modelados suficientemente consistentes. Realizando esta operación se obtienen los siguientes resultados:

```
# Get the most frequent products by ProductId
top_products = df['ProductId'].value_counts().reset_index()
top_products.columns = ['ProductId', 'Count']

top_products.head(250)
```

	ProductId	Count
0	B007JFMH8M	913
1	B002QWHJOU	632
2	B002QWP89S	632
3	B0026RQTGE	632
4	B002QWP8H0	632
...	...	...
245	B002LN1ME2	219
246	B000WVY4PE	219
247	B004E4HUMY	218
248	B006WYSFZK	218
249	B004728MI4	218

250 rows × 2 columns

Figura 3. Selección del subconjunto de datos para operar en el resto del proyecto

Con esta lista, se crea un array con los identificadores de estos 250 productos con el fin de crear un nuevo dataframe donde solo van a estar incluidos estos productos, de tal forma que se obtiene el resultado de la figura 4.

```
# Lista de los ProductId que deseamos obtener:
product_ids = ['B007JFMH8M', 'B0026RQTGE', 'B002QWHJOU', 'B002QWP89S','B002QWP8H0','B003B3OOPA','B001EO5Q64','B000VK8AVK','B0026KNQSA','B007M83302','B0

# Filtramos el DataFrame con los ProductId que deseamos obtener
df_patch = df[df['ProductId'].isin(product_ids)]

# Mostramos las primeras lineas del DataFrame filtrado
df_patch.head(10)

# Lo guardamos en un CSV
# filtered_df.to_csv('filtered_reviews.csv', index=False, encoding='utf-8')
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
20982	20983	B002QWP89S	A21U4DR8M6I9QN	K. M Merrill "justine"	1	1	5	1318896000	addictive! but works f
20983	20984	B002QWP89S	A17TDUBB4Z1PEC	jaded_green	1	1	5	1318550400	gen
20984	20985	B002QWP89S	ABQH3WAWMSMBH	tenisbrat87	1	1	5	1317168000	Pe
20985	20986	B002QWP89S	AVTY5M74VA1BJ	tarotqueen	1	1	5	1316822400	
20986	20987	B002QWP89S	A13TNN54ZEAUB1	dcz2221	1	1	5	1316736000	
20987	20988	B002QWP89S	A1AO8A38JUDVAF	chrisdd	1	1	5	1315180800	Magic Br
20988	20989	B002QWP89S	AH261OMJCV5U8	Chelle "Wichita, Kansas"	1	1	5	1314316800	
20989	20990	B002QWP89S	A3VVIU4JE56U4K	P. Chevalier	1	1	5	1314316800	
20990	20991	B002QWP89S	A3MWZ2E112KIY4	J. Nakawatase	1	1	5	1313884800	
20991	20992	B002QWP89S	A2PRQXWZRHTKE3	Maria Karlsson	1	1	5	1313193600	Great treat a

Figura 4. Creación del dataframe de trabajo

Se aprecia en la figura 4, cómo se crea dicho array y a continuación del dataframe se obtienen las filas que se corresponden con los identificadores de esos productos, que posteriormente haciendo una operación de *head* se obtiene un pequeño resumen de los productos que se han obtenido en este nuevo dataframe. Con la consecución de este dataframe ya se puede empezar a realizar las siguientes fases que son la descripción de fuentes de datos, y posteriormente la preparación y limpieza de los datos para poder crear el dataframe final con el que se realizará el modelado.

Descripción de los datos

Dentro del conjunto de datos del que se dispone, se pueden encontrar los siguientes campos descritos a continuación:

Campo	Valor	Tipo de dato

Id	Identificador único de la reseña	int64
ProductId	Identificador del producto asociado a la reseña	object
UserId	Identificador del usuario que realizó la reseña	object
ProfileName	Nombre del usuario que realizó la reseña	object
HelpfulnessNumerator	Número de usuarios que encontraron útil la reseña	int64
HelpfulnessDenominator	Número total de usuarios que calificaron la utilidad de la reseña	int64
Score	Puntuación dada al producto, escalada de 1 (menos puntuación) a 5 (de mayor puntuación)	int64
Time	Marca de tiempo en la que se realizó la reseña, en formato Unix	int64
Summary	Resumen o título breve de la reseña	object
Text	Texto completo de la reseña	object

Tabla 1. Resumen de características del dataset

Donde *object* representa las variables categóricas, una cadena, e *int64* son un numeros enteros. De momento las variables cuantitativas que se aprecian, tienen un tipo de dato acorde con lo que se espera, es decir, un número entero porque es *int64* y está bien. *ProfileName* es una cadena de texto pues es un nombre, *ProductId* e *UserId* se tratan de cadenas puesto que son un conjunto de caracteres alfanuméricos. *HelpfulnessDenominator*, *HelpfulnessNumerator* y *Score* representan valores enteros, ya que *HelpfulnessDenominator* es la utilidad del total de reseñas escritas y *HelpfulnessNumerator*, indica el número de reseñas que fueron útiles. Posteriormente, se tiene la puntuación (*Score*) que, en este caso, también es un número entero porque va de 1 a 5 con lo cual hasta ahora todo concuerda. Se aprecia también que *Summary* y *Text* son también cadenas, ya que representan por un lado el título de la reseña y por otro lo que es la reseña en sí, es decir, el texto completo. Otra cosa es la variable *Time* que se representa en formato Unix y en este caso es *int64* que es correcto, pero en este caso sí que se debería de convertir a un formato de fecha más legible, por tanto, aquí sí que habría que realizar una transformación.

Selección de variables

Antes de realizar las transformaciones oportunas de los datos y proceder al análisis univariante, bivariante, detección de valores nulos… se debe de tener en cuenta las variables que se van a necesitar para el estudio, ya que hay algunas que no harán falta para el objetivo que hay propuesto en este trabajo. El objetivo es clasificar los productos en función de las reseñas de los clientes de Amazon, por tanto, se debe tener en cuenta que las principales variables que se van a necesitar para nuestro estudio son, por un lado, el texto de la reseña y por otro, la valoración de los usuarios, que sería la variable objetivo. Es cierto que el resumen también da información, pero en este caso una de las variables principales es el texto de la reseña, ya que contiene toda la información necesaria para poder describir si la reseña ha sido buena o ha sido mala y en ese caso inferir si el producto es bueno o es malo.

Por otra parte, se tiene la puntuación o *Score*, que en este caso sería la variable objetivo puesto se pretende clasificar en base a si la puntuación es buena o mala. En este caso, sí va a ser

necesaria para el análisis ya que se desea asociar la puntuación de la reseña a la propia reseña, para que el modelo pueda aprender si es una buena reseña o es una mala, por tanto, en este caso es un campo fundamental. Además, se va a necesitar el identificador del producto, esto es importante sobre todo en este caso, ya que, se va a utilizar un subconjunto del total de datos y además este identificador servirá para que, en la partición de entrenamiento y test, se pueda realizar de una manera particular, ya que, habitualmente la separación en conjuntos de entrenamiento y test se realiza de manera aleatoria. En este caso también será así, sin embargo, se separará de manera aleatoria por conjunto de productos, esto es que todas las reseñas de un producto estarán o bien en el conjunto de entrenamiento o bien en el conjunto de test, pero en ningún caso habrá reseñas de un mismo producto en los dos conjuntos. Esta es la idea con esta variable.

El resto de campos no van a ser del todo útiles para el análisis a realizar en este proyecto, si bien es cierto que algunos de ellos cómo por ejemplo, *HelpfulnessNumerator* o *HelpfulnessDenominator*, han servido anteriormente para realizar ciertas estadísticas y comprobar la importancia de las reseñas de los productos de Amazon a la hora de comprender el negocio, para el análisis que se desea desarrollar no van a ser del todo útiles ya que no van a proporcionar ningún tipo de información relevante para discernir si el producto es bueno o es malo. Por tanto, según la metodología CRISP-DM, y como algunos autores como Carney (2020) y Consoy (2023) explican, antes de proceder a todo tipo de análisis sobre los datos, se pueden descartar aquellas columnas que no son relevantes para el objetivo del estudio siempre y cuando esté bien justificado, y se disponga de un buen análisis del negocio. Esto ahorra tiempo de procesamiento y recursos durante el proyecto.

Entonces, lo primero que se debe hacer es justificar cada una de las eliminaciones de cada una de estas columnas y por qué no son necesarias para el estudio que se va a realizar:

- *Id*: Se descarta porque es un identificador único para cada fila del dataset. No contiene información que influya en el contenido del texto ni aporta valor para la clasificación de productos. Teniendo en cuenta la metodología CRISP-DM, en la fase de preparación de datos, esta columna no es relevante para los objetivos que se han descrito en la fase de comprensión del negocio, ya que no ayuda a establecer la relación entre los textos y los productos.
- *UserId*: La identidad del usuario no influye en el contenido de la reseña ni en la clasificación de productos. Por tanto, según CRISP-DM, esta columna puede eliminarse ya que no tiene un rol claro en el objetivo de clasificación de los productos, definido durante la comprensión del negocio. Es irrelevante para el modelado basado en contenido del texto de las reseñas.
- *ProfileName*: El nombre de usuario tampoco aporta información semántica sobre el texto o el producto, no influye en la tarea de NLP. Durante la fase de preparación de datos, esta columna se puede identificar como ruido ya que mantenerla aumentaría la dimensionalidad de los datos innecesariamente, lo que es contrario al objetivo de optimización en esta fase.
- *HelpfulnessNumerator*, *HelpfulnessDenominator*: Aunque son relevantes para medir la calidad de las opiniones y pueden ser de utilidad en la comprensión del negocio, no son importantes para la clasificación de productos basada en texto. Por tanto, durante la comprensión de los datos, se identifican como métricas adicionales que no afectan directamente al contenido de los textos de las reseñas. Su inclusión podría añadir complejidad al modelo sin aportar beneficios significativos.

- *Time*: Es una marca temporal de cuando se realizó la reseña. Aunque puede ser útil, por ejemplo, en análisis de tendencias temporales, no tiene impacto en la clasificación de productos basada en texto. Durante la fase de preparación de datos, esta columna será descartada como irrelevante para el propósito definido en la comprensión del negocio. Mantenerla no aporta valor a la tarea de clasificación.

En cuanto al resto de características, sí se mantienen debido a las siguientes razones:

- *ProductId*: Identifica los productos y es fundamental para asociar las reseñas con los productos correspondientes. Esto es importante debido a la evaluación de los productos en cuanto a la etapa de modelado, y poder evaluar el rendimiento de un producto concreto en los test. Aunque podría ser eliminable, en este caso concreto es útil mantenerla para conservar la identidad de las reseñas referidas a cada producto.
- *Score*: Se mantiene dado que en el modelado permite asociar las reseñas positivas a valores de puntuación altos (Score con valores 4 o 5), las reseñas negativas a valores puntuación bajos (Score con valores 1 o 2) o valores neutros (Score con valor a 3). Además, permite ver la correlación entre los textos positivos o negativos en base a la puntuación y permite plantear los problemas de clasificación como supervisados, por lo que es muy interesante el enfoque.
- *Summary* y *Text*: Son la base del análisis NLP, ya que contienen el contenido textual necesario para clasificar los productos. Dado que son dos columnas o variables, se verá más adelante la estrategia que se va a seguir a la hora de tratarlas en el análisis. A priori, la información principal del análisis, la tendrá la variable Text, pero en Summary se puede encontrar información que puede ser valiosa a la hora de realizar el análisis y llegar a conclusiones importantes.

Transformación de los textos

En primer lugar, se deben eliminar aquellas columnas que no van a ser útiles a la hora de realizar el análisis, y que ya se ha justificado su eliminación el apartado anterior. Para ello, como se aprecia en la Figura 5 se seleccionan las columnas que se desean eliminar, se hace uso de la función *drop* y finalmente se muestra el resultado. Con esta operación, queda un subconjunto de datos de 86.565 filas y cuatro columnas, que será con el que se va a trabajar durante el resto del proyecto.

Esto nos deja un conjunto de datos aproximadamente del 15% del conjunto de datos original qué es un volumen razonablemente bueno para poder ejecutar el análisis con solvencia. Dado el gran volumen de datos del conjunto original, es importante disponer de la mayor cantidad de datos posibles sin comprometer el rendimiento de la versión gratuita de Colab. Con esto se asegura un compromiso entre la mayor cantidad de datos posibles que se puedan utilizar sin comprometer el funcionamiento de la plataforma en su versión gratuita ya que si no se debería pagar por el servicio para obtener un mejor rendimiento.

```
# Seleccionamos las columnas que deseamos eliminar
columns_drop = ['Id', 'UserId', 'ProfileName', 'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Time']

# Eliminamos las columnas seleccionadas
df_tiny = df_sample.drop(columns=columns_drop)

# Mostramos el resultado para comprobar la eliminación
df_tiny.head()
```

	ProductId	Score	Summary	Text
0	B002QWP89S	5	addictive! but works for night coughing in dogs	my 12 year old sheltie has chronic brochotitis...
1	B002QWP89S	5	genuine Greenies best price	These are genuine Greenies product, not a knoc...
2	B002QWP89S	5	Perfect for our little doggies	Our dogs love Greenies, but of course, which d...
3	B002QWP89S	5	dogs love greenies	What can I say, dogs love greenies. They begg ...
4	B002QWP89S	5	Greenies review	This review is for a box of Greenies Lite for ...

Pasos siguientes: Generar código con df_tiny | Ver gráficos recomendados | New interactive sheet

Figura 5. Eliminación de las columnas del dataframe no relevantes para el estudio

Para poder realizar un estudio de las características de los textos lo primero que se debe hacer es realizar una transformación de los mismos. Para ello, se irán comentando las operaciones necesarias para poder realizar la transformación de textos que va a permitir hacer el análisis univariante, bivariante…

En el caso de este estudio, se va a necesitar tokenizar el texto y para ello se deberá tokenizar al nivel de palabras ya que, este es el utilizado en la mayoría de modelos NLP, pero sobre todo es usado en análisis de sentimientos o clasificación que es el propósito principal de este proyecto. Para ello, el primer paso es eliminar los caracteres especiales como pueden ser URLS menciones hashtags o cualquier tipo de carácter que no tenga que ver con las palabras en sí, qué son las que contienen la semántica necesaria para hacer la clasificación o el análisis de sentimientos. Esta eliminación ayuda a quitar ruido de los datos, mejorando la calidad del texto como se acaba de indicar y hace que los algoritmos trabajen con información más importante. También ayuda a reducir el nivel de vocabulario a utilizar en el modelo y esto hace que el modelo sea más eficiente y tenga que lidiar con menor número de características lo que hace que el análisis sea más limpio. Por último, también mejora la tokenización ya que evita que en la etapa de división del texto en partes más pequeñas se confundan los caracteres con palabras.

La conversión de los caracteres mayúsculas a minúsculas es importante ya que ayuda a eliminar ambigüedades, esto es porque una misma palabra puede ser escrita con mayúsculas, minúsculas e incluso, la primera con mayúscula y el resto en minúsculas. El significado de la palabra es idéntico pero el algoritmo podría interpretar que son distintas, por tanto, es necesario pasarlas a un formato uniforme y éste es en minúsculas. Por ende, esto también ayuda a reducir la dimensionalidad del vocabulario ya que, si una misma palabra que para un humano tiene el mismo significado, pero para un algoritmo dependiendo de si está escrito con mayúsculas o minúsculas lo entiende de manera distinta, ayuda que, si se tiene una representación uniforme, hace que a la hora de la codificación se reduzca el número de posibilidades a una única por cada palabra. Esto también ayuda a la hora de realizar los entrenamientos con técnicas NLP.

También es importante la eliminación del uso de las *stopwords* y es que se trata de palabras que no tienen significado por sí mismas, sino que aportan valor de comprensión en un texto cómo pueden ser preposiciones, artículos… Posteriormente a estas etapas, se realiza la tokenización que es la división del texto en partes más pequeñas, y como se ha indicado antes, en el caso del análisis de sentimientos o clasificación de textos cómo es el propósito de este proyecto, hay que hacerlo al nivel de palabras, ya que, se puede tokenizar al nivel de oraciones más pequeñas o de párrafos más pequeños. Sin embargo, para este proyecto es importante llegar hasta el último nivel de granularidad. Hay distintas librerías en Python útiles para este propósito cómo *NLTK*,

SpaCy, *TextBlob*… En este caso, se hará uso de la librería *SpaCy* ya que es sencilla de usar y ya ha sido vista con anterioridad a lo largo del presente máster de Data Science.

Ya en el proceso de transformación, en un primer paso, se procede examinar si los números son necesarios convertirlos a su equivalente en palabra. Esto es importante ya que puede darse el caso de que los números sean significativos en el contexto, o la semántica del texto, por tanto, hay que cerciorarse de que si se eliminan no se pierde información relevante a la hora de la clasificación del texto.

Si se realiza una inspección sobre los primeros campos tanto del sumario como del texto, se pueden encontrar referencias a la edad de un animal, o al peso de un animal para poder expresar la opinión respecto a un producto por lo que, puede ser importante en este caso dejar los números y pasarlos a su equivalente en palabra escrita ya que, puede ser más rico semánticamente a la hora de analizar el texto. Tomando este punto de vista, no se pierde información y en la etapa de *stopwords* con las librerías qué están implicadas en ese proceso, se determinará con los paquetes de Python necesarios, qué números y cuáles no se consideran *stopwords.*

Una vez determinada la necesidad de convertir los números a su equivalente en palabra escrita, se realizan las siguientes operaciones en las columnas *Summary* y *Text.* Para ello se utiliza la librería *Inflect* cómo se ha estudiado y documentado en artículos como el de Elgort B. (2022), de lo que pudiera parecer.

En primer lugar, como se aprecia en la Figura 6, se toman las columnas del dataframe que se deben transformar (*Summary* y *Text*), y se le aplica la función lambda a cada elemento de la columna. Mediante el paquete re de Python, se elimina todo lo que no aparezca en la expresión regular denotada por *r'[^a-zA-Z\s]',* esto incluye caracteres extraños, signos de puntuación, corchetes, paréntesis… Con la función *join*, se une las palabras de nuevo en una sola cadena separadas por un espacio. Posteriormente se utiliza *inflect_engine* para transformar los números en palabra escrita. En este punto, se realiza una comprobación con una sentencia *if* para determinar si es un número, donde si es un número lo convierte a palabra escrita, y si no es un número lo deja como está. Mediante un bucle, se va realizando esta operación sobre todas las palabras que aparecen para cada par de columna fila referente a *Summary* y *Text*, es decir, cada celda que contiene texto. Notar que al realizar estas transformaciones se crea una nueva columna en el dataset llamado *Transformed_Summary* y *Transformed_Text* con el objetivo de comprobar cómo se han realizado las transformaciones y si han sido de forma satisfactoria.

```
df_tiny['Transformed_Summary'] = df_tiny['Summary'].apply(lambda text: re.sub(r'[^a-zA-Z\s]', '', ' '.join(
        inflect_engine.number_to_words(word) if word.isdigit() else word for word in text.split()
    )))
df_tiny['Transformed_Text'] = df_tiny['Text'].apply(lambda text: re.sub(r'[^a-zA-Z\s]', '', ' '.join(
        inflect_engine.number_to_words(word) if word.isdigit() else word for word in text.split()
    )))
```

Figura 6. eliminación de caracteres extraños

En esta primera instancia, ya se tienen eliminados tanto los caracteres extraños cómo convertidos los números a letra escrita. Después, si los números son considerados como s*topwords* en la siguiente etapa que se describirá ahora, se eliminarán si la librería lo considera preciso.

A continuación, se va a describir en una segunda fase que se muestra en la Figura 7, donde se ilustran las operaciones que se deben de realizar para Tokenizar, Lemanizar, poner letras

mayúsculas como minúsculas… en el texto. En este caso se parte de las columnas *Transformed_Summary* y *Transformed_Text* y sobre ellas se realizan las operaciones pertinentes. De nuevo, se aplica una función lambda para poder procesar el texto de cada fila. En dicha función se aplica la función *nlp* qué es un modelo de procesamiento de lenguaje natural cargado con *SpaCy* y procesa el texto y devuelve un objeto de tipo documento que contiene una lista de tokens con palabras, puntuaciones… Junto con información adicional como su lema, etiqueta, gramatical, si es una *stopword*… Es importante, comentar que para poder tokenizar adecuadamente, se debe de cargar el modelo de SpaCy para la tokenización. En este caso debe ser en inglés pues es el idioma en el que están las reseñas, y además se selecciona un modelo de pocas palabras para ahorrar recursos. Esto se hace mediante la instrucción *nlp = spacy.load('en_core_web_sm'),* antes de realizar la tokenización.

```
df_tiny['Transformed_Summary'] = df_tiny['Transformed_Summary'].apply(lambda text: " ".join(token.lemma_.lower()
        for token in nlp(text) if not token.is_stop and not token.is_punct))
df_tiny['Transformed_Text'] = df_tiny['Transformed_Text'].apply(
    lambda text: " ".join(token.lemma_.lower()
        for token in nlp(text) if not token.is_stop and not token.is_punct))
```

Figura 7. Tokenización de las columnas de texto

Esto se va realizando en un bucle *for* para recorrer cada palabra del texto procesado, y con una sentencia *if* se va realizando un filtro de tokens e irrelevantes, excluyendo *stopwords* (*token.is_stop*), signos de puntuación (*token.is_punct*) y por último también obtiene el lema del token (*token.lemma_*). Además, con la función *lower* se colocan las letras mayúsculas en minúsculas. Todo esto se aplica con la función *join* para qué se haga una Unión de todas las palabras en una sola cadena separadas por un espacio.

El resultado de aplicar estas transformaciones se puede ver de manera clara en la Figura 8. Se puede apreciar como los tokens irrelevantes han desaparecido, las letras mayúsculas ahora son minúsculas, los números los ha considerado como *stopwords* y por tanto han sido eliminados… se puede apreciar que el resultado aparece en texto plano no aparece en los textos tokenizados. Esto tiene una razón de ser, ya que dependiendo de los modelos de aprendizaje automático que se vayan a aplicar como se explicará y se verá detalladamente más adelante en algunos casos es necesario transformar el texto a vectores mediante técnicas como TF-IDF, y en otros casos el algoritmo de aprendizaje automático (o la librería), es capaz de gestionar el texto plano y se encarga de la tokenización

Summary	Text	Transformed_Summary	Transformed_Text
yummy	These chips are wonderful. My kids and I all ...	yummy	chip wonderful kid love salt pepperbr oz bag...
Great Item	Great packaging, and item is just as described...	great item	great packaging item describe purchase item sh...
Excellent chips	Just to give a little background - I ordered t...	excellent chip	little background order great deal box singl...
Perfect for WW followers	Got this item on a one-day-only sale on Amazon...	perfect ww follower	get item onedayonly sale amazon love have sing...
Variety pack, packed lunches	I discovered Pop Chips awhile back and fell in...	variety pack pack lunch	discover pop chips awhile fall love bbq flavor...
Loved all the flavor except salt and pepper!	I really liked these chips as did my entire fa...	love flavor salt pepper	like chip entire family wish pick choose flavo...
chips	These popchips were something I wanted to try ...	chip	popchip want try bake barbeque one okay rest t...
Good as a sampler pack of the various flavors	I like these Pop Chips a lot, but not all the ...	good sampler pack flavor	like pop chips lot flavor variety pack make go...

Figura 8. Resultados de las operaciones de Tokenización, lemanización...

Análisis univariante

Ahora se reutiliza, aunque adaptado, una parte de código ya utilizada en uno de los entregables evaluables, concretamente del módulo 7, dónde nos permite clasificar las variables en dos dataframes por un lado las variables cualitativas y por otra las variables cuantitativas. También se calculan los estadísticos básicos como la media, el valor mínimo, el valor máximo y la mediana,

en el caso de las variables cuantitativas. En el caso de las cualitativas se calcula el tipo de variable y el número de clases. Con esto, se puede obtener una primera visión de las variables.

En cuanto a las variables cualitativas se tienen tres que son *ProductId, Summary y Text*. Si se presta atención al número de clases qué hay en *ProductId* es de 250 correspondiente a los 250 productos que se han seleccionado del dataset original. Sin embargo, en las columnas *Summary* y *Text* se aprecia cómo aparece un gran número de clases, esto es absolutamente normal ya que cada texto de resumen es distinto y cada reseña es distinta.

A continuación, se profundizará un poco más en el análisis y se calculará para estas 3 variables cualitativas distintas estadísticas. Para ello se aprecia la Figura 9, donde en el caso de *ProductId,* se calcula el número de valores únicos obteniendo 250, algo lógico porque ya se ha calculado anteriormente y es el número distinto de productos que hay en el conjunto de datos, también se calcula el valor más frecuente y sale el identificador del producto del cual existen más reseñas con un valor de 913.

```
Estadísticas ProductId:

Valores únicos: 250
Valor más frecuente: B007JFMH8M
Frecuencia del valor más frecuente: 913


Estadísticas de Summary:

Valores únicos: 30538
Longitud media (caracteres): 23.972124992779992
Mayor longitud (caracteres): 128
Menor longitud (caracteres): 2


Estadísticas de Text:

Valores únicos: 36370
Longitud media (caracteres): 475.6673944434818
Mayor longitud (caracteres): 21409
Menor longitud (caracteres): 18


Estadísticas de Transformed_Summary:

Valores únicos: 24062
Longitud media (caracteres): 17.107983617398972
Mayor longitud (caracteres): 109.0
Menor longitud (caracteres): 1.0


Estadísticas de Transformed_Text:

Valores únicos: 36350
Longitud media (caracteres): 250.58993819673077
Mayor longitud (caracteres): 13177
Menor longitud (caracteres): 9
```

Figura 9. Estadísticos básicos de las variables de texto

En el caso de la variable *Summary*, calculando el número de valores únicos sale alrededor de 30538 donde la longitud media de los caracteres escritos es en torno a 24 caracteres, el caso de la mayor longitud del resumen son unos 128 caracteres y en el qué menor se encuentran 2 caracteres. Si se realiza lo mismo para la variable *Text*, hay un número de valores únicos de 36370, la longitud media de la reseña es en torno a 475 caracteres algo que dice que las reseñas son bastante más extensas que el resumen y se muestra que la mayor longitud de una reseña son 21409 caracteres aproximadamente con una menor longitud de 18 caracteres. Esto ya da bastante información, sobre todo en el caso de las variables de resumen y texto. Si se aprecia el número de valores únicos, es ligeramente distinto, pero ya indica que puede haber diferencias en el caso de estas dos variables. Destacar que, en el caso de las variables de texto ya transformadas, estás estadísticas como por ejemplo la longitud mayor o longitud media son menores debido a la eliminación de palabras como *stopwords*.

Si se atiende a la lógica, se puede ver que las características de texto que hay en el resumen, puede no haberlas en el texto y viceversa, es decir, puede haber palabras que puedan significar lo mismo, pero que puedan ser sinónimos, o la expresión puede ser distinta… pero que puede haber relación entre estas dos variables. Esto hace pensar que más adelante se podría crear una *feature* que contenga las palabras existentes en estas dos columnas, con el fin de dotar al análisis de mayor riqueza sintáctica. El significado del resumen y del texto van de la mano, solo difiere en la forma de expresión en una columna de una manera más corta (*Summary*), profundizando más en el caso de la columna *Text*.

Sí se grafican los valores de estas variables, se obtienen las siguientes gráficas que se aprecian en la Figura 10, Figura 11 y Figura 12. En el caso de cada producto se aprecia lo que ya se obtuvo cuándo se seleccionaron los productos que van a conformar parte del subconjunto de datos que se está utilizando para el desarrollo de este proyecto. Se aprecia que hay una distribución de reseñas desigual, donde un producto tiene una alta concentración de reseñas y luego hay distintos productos que sí que mantienen un número de reseñas parejo. Esto puede decir que hay productos que son más populares entre los compradores, ya que eso hace que se generen más reseñas.

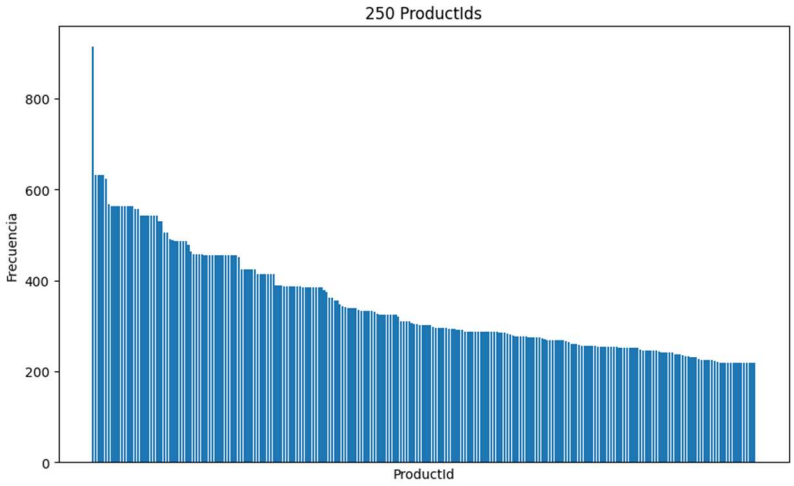


Figura 10. Distribución de reseñas por producto

Si se analiza la variable *Summary* en la Figura 11, como ya se ha apreciado, la mayoría de los resúmenes están en torno a 5 y 35 caracteres (de media eran 24), pero no hay un gran número de resúmenes extremadamente largos, como de resúmenes extremadamente cortos. Esto dice que los usuarios tienden a dejar descripciones breves y concisas. Esto puede ser útil para complementarse en el análisis con el texto completo de la reseña.

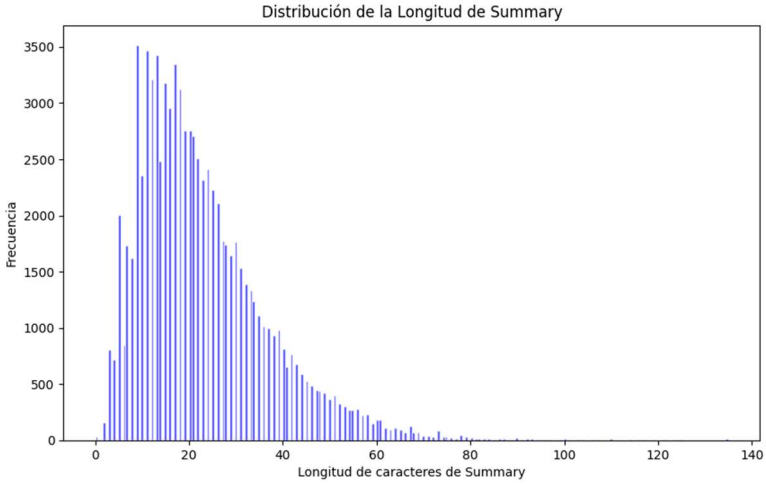


Figura 11. Distribución de longitud de caracteres en la variable Summary

Finalmente, si se realiza un análisis de los textos completos de cada reseña, se aprecia que estos tienen una distribución de caracteres más amplia que en el caso de los resúmenes y que lo más normal es que las reseñas tengan una longitud en torno a 200 o 500 caracteres. Es cierto también que, aunque no son los mayores casos, se pueden encontrar textos muy largos de en torno a 1500 o 2000 caracteres, o incluso llegando a los 3000-4000 caracteres, pero son casos excepcionales. Como se ha comentado anteriormente, los textos completos ofrecen una mayor riqueza semántica debido a que son mucho más largos que los resúmenes.

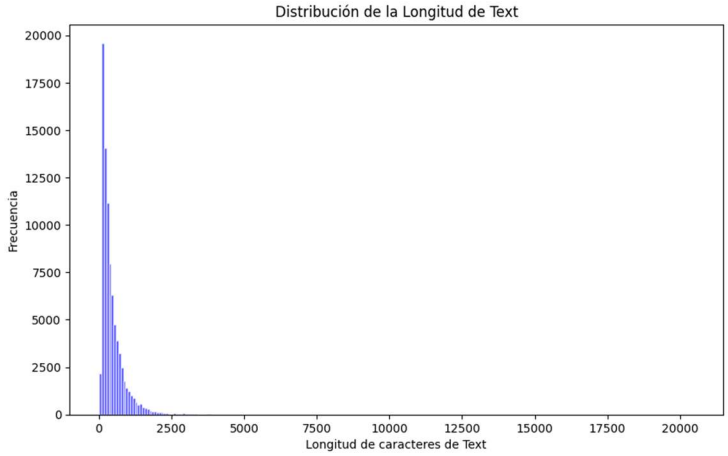


Figura 12. Distribución de longitud de caracteres en la variable Text

Se podría ir un poco más allá en el análisis, y obtener las palabras que se ven con mayor frecuencia tanto en la columna *Transformed_Summary* y la columna *Transformed_Text*. Para ello se generan dos histogramas realizando un conteo de las palabras que aparecen con más frecuencia en cada columna de tal forma que se obtienen los siguientes resultados, mostrados en la Figura 13.

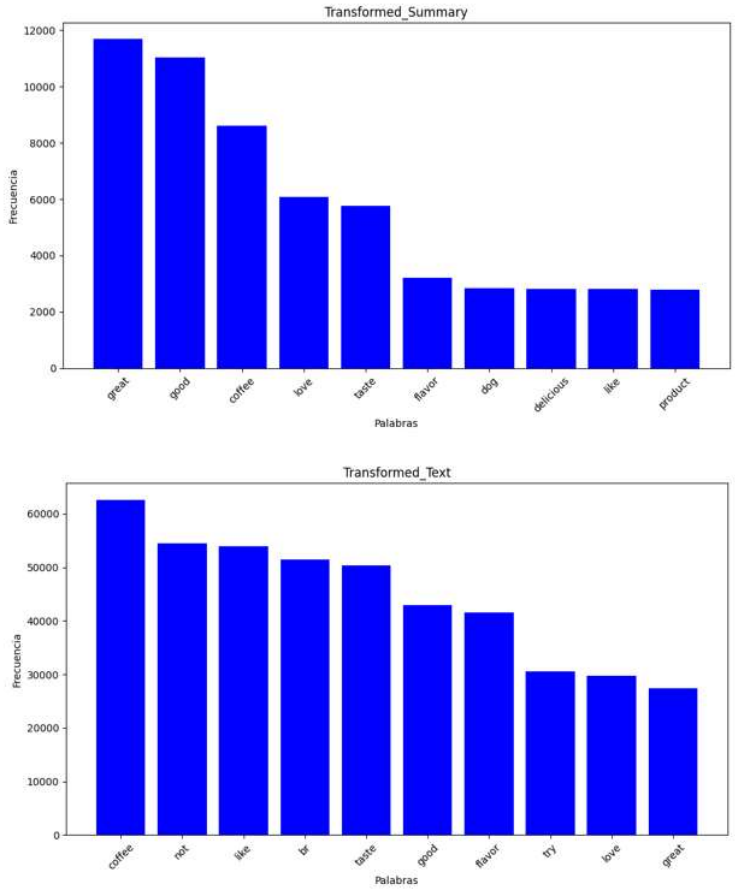


Figura 13. Palabras con mayor frecuencia por columna

Se observa, que aparecen palabras de buen grado cómo puede ser *"great, good, love…",* y alguna de mal grado como "*not*". Sin embargo, esto es algo que se debe de analizar en el análisis bivariante ya que, se puede disgregar la frecuencia de las palabras en cada columna de textos, pero asociada a su vez a cada valor de puntuación y en ese caso se obtendrán las palabras más frecuentes en caso de que la puntuación sea alta o sea baja. Por tanto, en este caso lo que se ve es un agregado de las palabras con mayor frecuencia que aparecen en el texto. En general nos puede dar una información por ejemplo del tipo de productos que se tienen en este conjunto de datos como es el café o comida para perros.

Pasando al análisis de la variable *Score,* apreciando la Figura 14, en este caso se trata de una variable cuantitativa de tipo discreto ya que toma valores dentro de un rango determinado entre 1 y 5 y siempre valores enteros se aprecia también que la media es 4,16 es decir una media bastante elevada lo que hace pensar que las reseñas que se van a tener en el conjunto son de buenas opiniones. Esto es sobre todo lo que se aprecia en la mediana que es de valor 5.

	Nombre	Tipo de variable	Media	Mínimo	Máximo	Mediana
0	Score	Cuantitativa discreta	4.167735	1	5	5.0

Figura 14. Estadísticos básicos de la variable Score

Si se analiza gráficamente la variable, se puede apreciar que efectivamente que la mayor parte de las reseñas son buenas o muy buenas, estando la mayoría en el rango de las que tienen una puntuación de 4 o 5. Por tanto, en esta variable se tendrá que realizar una compensación, para que el conjunto de datos sea más balanceado. Por tanto, habrá que aplicar técnicas de

submuestreo en aquellas reseñas que estén con puntuación de 4 o 5, o bien realizar técnicas de sobremuestreo o SMOTE para poder incrementar aquellas muestras de las reseñas que contienen valores de puntuación de 1, 2 o 3. Esto se verá un poco más adelante, donde se realizará el análisis y se aplicará la técnica más adecuada.

Esto es algo común en todo el dataset ya que, si se analizan las 568000 filas de las que está compuesto el dataset original, y se realiza este mismo análisis de la variable *Score* como se puede ver en la Figura 15, se aprecia cómo predominan las reseñas con puntuaciones altas siendo la distribución muy parecida a la que se tiene gráficamente en este apartado. Por tanto, si se desease aumentar el número de muestras razonando que se podría obtener más reseñas que tienen puntuaciones 1, 2 o 3, se seguiría obteniendo una distribución de valores en la variable score como la que se ha obtenido en este punto. Esto se debe a que se está realizando este estudio estrictamente con determinados productos, ya que si no fuera así se podría seleccionar puntuaciones aleatoriamente de otros productos que no han sido seleccionados en este subconjunto de 250 productos, aplicando a las clases que tienen menor puntuación reseñas de más productos más allá de los 250 que se han seleccionado.

Por ejemplo, si se realiza una suposición en la que se tuviera este subconjunto pero que para poder balancearlo en las clases 1, 2 o 3, se seleccionan reseñas de más productos de los contenidos en el dataset original, es decir, que se pudiera dar el caso que para estos valores de puntuación se tuvieran reseñas de 60 productos distintos, pero que en el caso de los que tienen puntuación más alta quedan exactamente igual. Esto podría ser un enfoque válido ya que se aumentarían el número de muestras de las clases 1, 2 o 3 con mayor riqueza lingüística. El problema es que ya no se estaría analizando con determinados productos, ya que, si esto se extrapolara a todo el conjunto de datos con las 568000 filas, y se realizara todo el análisis con estas 568000 filas, se tendría el mismo problema solo que no se podría proceder del mismo modo ya que no habría más productos de los que obtener reseñas negativas para poder balancear el conjunto de datos. Por tanto, aunque sería una opción realizarlo de esta manera no es del todo sensato.

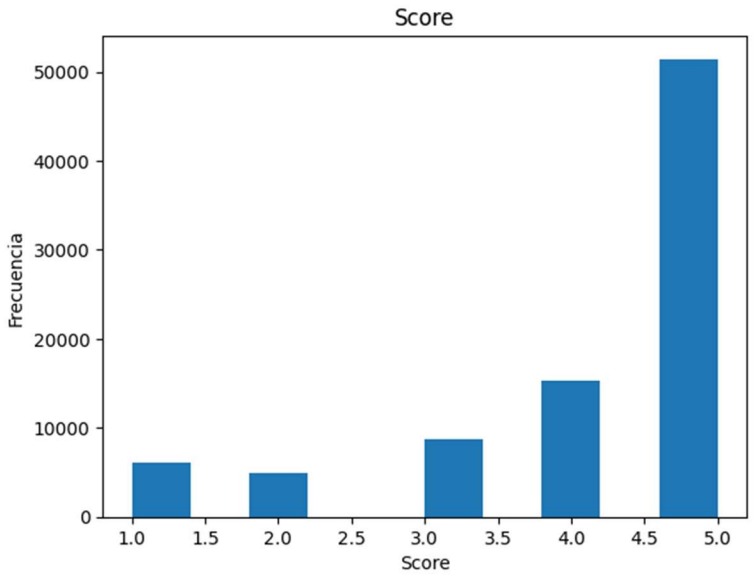


Figura 15. Distribución de las reseñas por la variable Score

Entonces, la mejor técnica para abordar este problema y poder balancear más el conjunto de datos, sería realizar un sobre muestreo aumentando el número de muestras de aquellas reseñas

de las que se dispone de pocas muestras, es decir, de aquellas reseñas con puntuaciones 1, 2 o 3. De esta manera se consigue mantener en las clases que tienen mayor puntuación y por tanto mayor número de muestras, la riqueza lingüística, que es muy útil a la hora de proceder con análisis NLP. Esto es así, porque si se eliminan muestras de las clases que tienen mayor puntuación como pueden ser las de 4 o 5, se podría estar perdiendo riqueza lingüística que para el análisis NLP es muy útil.

Además de este problema de balanceo de datos, cabe destacar otro aspecto importante que no es otro que era la hora de la clasificación de textos, y por ende, la que va a permitir clasificar los productos en buenos o malos, se necesita no clasificar los productos en función de la puntuación obtenida, sino clasificarlos en función de si tienen una buena valoración o mala valoración. Esto hace pensar que se va a necesitar crear una *feature* que englobe las opiniones de 4 o 5 como buenas, las opiniones de 1 o 2 como malas y 3 como neutras. Por ello, en el razonamiento anterior se estaba hablando de aumentar las muestras de aquellas reseñas que tenían como puntuación 1, 2 o 3, ya que, si se toman aquellas reseñas que tienen como puntuación 4, son ligeramente superiores a las que tienen puntuación 3.

Análisis Bivariante

Ahora se va a realizar un análisis entre cada par de variables con el fin de encontrar relación entre las mismas. En lo que más se tiene que incidir en este caso es en la relación que tiene la variable *Score* con las variables que contienen los textos, puesto que en el apartado anterior se ha visto cómo se puede obtener frecuencias de palabras en base a cada puntuación y con ello poder relacionar la puntuación con las palabras que van apareciendo en dichas reseñas.

Si se realiza un análisis sobre estadísticos básicos del texto frente al identificador de producto, como se aprecia en la Figura 16, se ve como sí que se puede extraer alguna conclusión, y es que en las medias se va a reflejar la longitud en caracteres. En la longitud del texto, las reseñas completas van a tener una longitud mayor en caracteres que el caso de los resúmenes con lo cual no va a indicar demasiada información nueva. En el caso de la mediana, indica la longitud típica de las reseñas esto es útil si hay *outliers,* ya que, la mediana no es tan sensible a valores extremos como la media. El máximo y el mínimo de los textos, no va a indicar nada nuevo que no se sepa hasta ahora, ya que, se ha tratado con anterioridad la longitud, sin embargo, la desviación sí que indica la variabilidad de la longitud de las reseñas aquí los productos con valores altos tienen reseñas con longitudes muy variadas, mientras que los productos con valores bajos tienen reseñas más consistentes en longitud.

Para el caso de los resúmenes, es lógico que la media los valores máximos y mínimos sean menores que en el caso de los textos, la mediana indica que la mayoría de los productos tienen resúmenes de 2 o 3 palabras lo que podría reflejar que los usuarios suelen ser escuetos y concisos en los resúmenes, y la desviación indica la variabilidad de la longitud del resumen dónde se puede apreciar qué hay algunos con mayor desviación que otros.

Estadísticos Text vs ProductId:

	ProductId	Mean_Text	Median_text	Max_text	Min_text	Std_Dev_text
0	B0001ES9F8	26.260417	21.0	121	1	18.786483
1	B0007A0AQM	32.476415	23.0	308	5	29.961747
2	B0007A0AQW	32.476415	23.0	308	5	29.961747
3	B0009F3POY	42.397260	29.0	317	7	40.098184
4	B0009YJ4CW	53.712062	43.0	231	8	38.653039
..	...	...	...	...	...	...
245	B008RWUKXK	63.016949	50.0	322	8	49.258271
246	B008ZRKZSM	36.606452	28.0	462	7	36.181312
247	B0090X8IPM	59.024528	47.0	377	8	43.863482
248	B009E7YC54	37.445378	26.0	271	7	33.025293
249	B009RB4GO4	26.653386	20.0	113	6	20.544863

[250 rows x 6 columns]

Estadísticos Summary vs ProductId:

	ProductId	Mean_Sum	Median_Sum	Max_Sum	Min_Sum	Std_Dev_Sum
0	B0001ES9F8	2.690972	2.0	7	1	1.276287
1	B0007A0AQM	2.905660	3.0	9	1	1.354640
2	B0007A0AQW	2.905660	3.0	9	1	1.354640
3	B0009F3POY	2.337900	2.0	7	1	1.261635
4	B0009YJ4CW	2.754864	2.0	8	1	1.405286
..	...	...	...	...	...	...
245	B008RWUKXK	2.877966	3.0	8	1	1.456404
246	B008ZRKZSM	2.558065	2.0	8	0	1.490482
247	B0090X8IPM	3.088679	3.0	12	1	1.693327
248	B009E7YC54	2.542017	2.0	10	1	1.535899
249	B009RB4GO4	2.450199	2.0	8	1	1.475300

[250 rows x 6 columns]

Figura 16. Estadísticos básicos Text vs. ProductId

Sí que es importante mencionar que la diferencia entre el máximo del texto y su media puede dar lugar a la existencia de outliers con lo que habría que gestionar esta casuística. Si se tiene en cuenta que lo lógico sería eliminar dichos *outliers* para evitar la presencia de ruido que pudiera interferir en el análisis en los modelos, en este caso no sería del todo idóneo ya que si se eliminan los textos que tienen una longitud elevada también se estaría perdiendo mucha información semántica que para la clasificación de textos o análisis de sentimientos de los textos sería crucial más aún, cuando se están eliminando aquellos textos con mayor longitud y por tanto con mayor riqueza semántica por lo tanto en este caso no sería lo más idóneo eliminar los outliers.

Si se realiza un análisis gráfico se pueden tomar los primeros 20 productos como se puede apreciar en la Figura 17, y en este caso si se analiza el texto frente al identificador de producto, se puede apreciar cómo la mediana (que es la línea verde), varía significativamente entre productos algunos productos tienen textos más extensos mientras que otros tienen textos más cortos, el rango inter cuartil, qué es el ancho de las cajas, para la mayoría de productos es amplio lo que sugiere una variabilidad moderada de la longitud de las reseñas dentro del mismo producto, sin embargo, hay otros que lo tienen más estrecho aportando más consistencia en la longitud de sus reseñas. En cuanto a los outliers, la longitud máxima de texto de algunos productos es considerablemente mayor que en otros por lo que la dispersión puede ser mayor.

En el caso del resumen frente al identificador de producto, la mediana la longitud del resumen es consistente entre productos situándose entre 10 y 30 caracteres en la mayoría de los casos lo que indica que los usuarios suelen ser breves en los resúmenes. En cuanto al rango intercuartílico la mayoría de productos lo tienen relativamente estrecho lo que sugiere que los resúmenes son más uniformes en comparación con las reseñas completas, además algunos productos tienen un rango más amplio lo que indican una mayor variación entre resúmenes. Tampoco se observan puntos fuera de las líneas finas pero los productos con líneas finas más largos pueden tener valores atípicos dentro de los límites permitidos.

En definitiva, si se toman los resúmenes para el modelado y clasificación de las reseñas, hace que sea más homogéneo pero que se disponga de muchísima menos información, por lo que completos, es encarecidamente recomendable hacer uso de ellos.

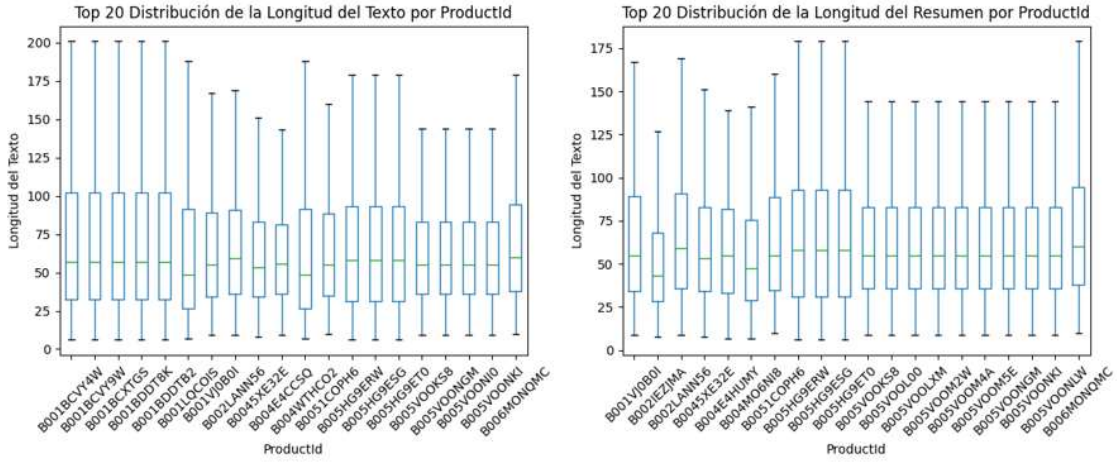


Figura 17. Distribuciones de longitud de Text y Summary por ProductId

Si se realiza un análisis entre las variables del identificador de producto y la puntuación de las reseñas, puede servir para ver cómo es la distribución de las valoraciones de las reseñas en cada producto. En la Figura 18, se puede observar que es coherente con el análisis de la distribución de las puntuaciones que se realizó en el análisis univariante, ya que en ese caso se realizó un estudio global de la distribución de las puntuaciones y se concluyó que el conjunto de datos estaba desbalanceado para poder clasificar las reseñas en base a sus puntuaciones. En este caso se puede ver con un grado más de detalle, ya que se puede observar qué es esto que se está describiendo es algo común a la mayoría de los productos y es que se dispone de reseñas con puntuaciones muy altas, es decir, de 5 puntos sobre 5 y siendo ampliamente menor en el resto de puntuaciones. Se puede observar en algunos productos como hay reseñas más negativas que en otros, y esto se puede apreciar ya que en la gráfica aparece en color azul. Sin embargo, son pequeños casos aislados en los que se puede ver esta casuística.

De manera general predomina aquellas reseñas donde la puntuación es la más alta, posteriormente es común en todos los productos o en la amplia mayoría de ellos una puntuación alta, aunque en una proporción notablemente menor que en las de puntuación de valor 5. Y finalmente se pueden apreciar que suele haber pocas puntuaciones bajas o medias, lo que hace indicar que en general los productos suelen tener una buena aceptación entre los usuarios. Sin embargo, puede ser mejorable ya que en algunos casos como se acaba de comentar, en algunos productos, aunque en la mayoría de sus reseñas tiene una puntuación muy alta se puede observar cómo existen un amplio número de reseñas muy negativas, de valor 1. Por tanto, este tipo de productos después del análisis son los que pueden aparecer más penalizados con respecto a otros que tengan un número de reseñas negativas mucho menor.

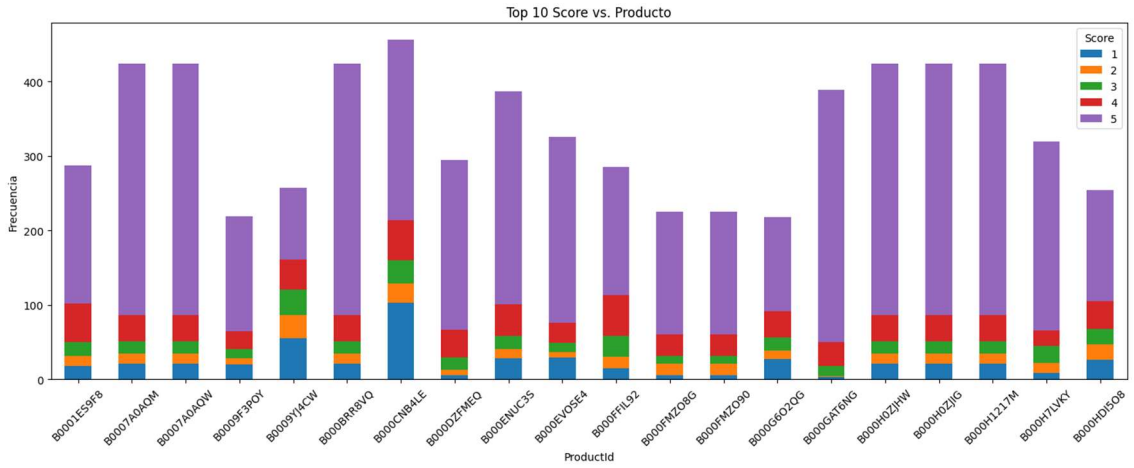


Figura 18. Distribución de reseñas por producto

Como ya se ha comentado anteriormente, es interesante analizar las variables que contienen los textos frente a la variable de puntuación teniendo en cuenta las puntuaciones que pueden ser positivas y negativas, es decir, serán reseñas negativas aquellas que tengan una puntuación de 1 o 2, serán neutras o se considerarán neutras aquellas que tengan una reseña de valor de puntuación 3, y serán positivas aquellas reseñas que contengan valores de puntuación 4 o 5. Por tanto en un primer análisis se realizarán histogramas atendiendo únicamente al valor de cada puntuación, es decir, se generarán 5 histogramas y se agruparán teniendo en cuenta el razonamiento que se acaba de describir.

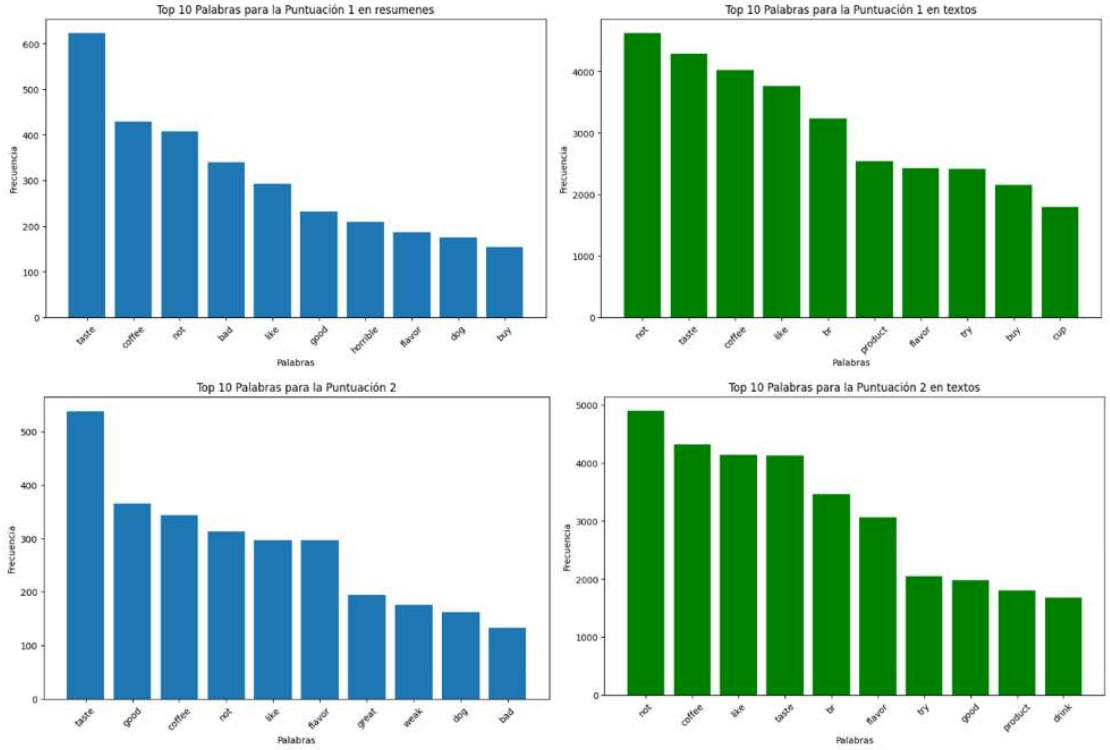


Figura 19. Mayor frecuencia de palabras para Summary y Text en función de la variable Score (Puntuación 1 y 2)

Al realizar estos histogramas frente a la variable de puntuación, como se aprecia en la Figura 19, en el caso de los resúmenes se puede apreciar la diferencia con los que se realizaron en el análisis un invariante y aquí sí se puede apreciar las palabras que aparecen con mayor frecuencia en

función de la puntuación de la reseña. Para el caso de los valores de puntuación 1 y 2, que son aquellos que se corresponden con reseñas negativas se puede apreciar qué hay palabras bastante descriptivas y que expresan la disconformidad de los usuarios frente a estos productos. Palabras como "*horrible, bad, not*" son frecuentes en este tipo de reseñas.

Es cierto que también aparecen otras palabras de buen grado como pueden ser "*Good, great*", y puede deberse porque a la hora de expresar una opinión pueden aparecer frases con sentido negativo en general, pero que aparezca con estas palabras dentro de esa frase. Otro de los aspectos a tener en cuenta, es que los usuarios pueden hacer uso de la ironía para marcar su desaprobación sobre ese producto, y esto es algo que en el análisis de sentimientos es difícil de captar, ya que analizando el texto palabra a palabra, no se puede percibir esta casuística. Se debe de tener en cuenta, que se pueden producir también errores humanos, es decir, usuarios que han confundido el sistema de puntuación asignando una puntuación 1 cómo que sea la mejor y no al revés, puede que en el proceso de transformación de los datos de texto se hayan perdido palabras clave o sufijos que dan contexto a la oración… Si se atiende a las frecuencias de estas palabras, se puede ver que son menos frecuentes que aquellas que expresan disconformidad por lo que en cómputo general se puede entender cómo que sí son negativas estas reseñas.

Para el caso de los textos, esta diferencia no se ve tan clara, ya que, se puede apreciar una única palabra de disconformidad que es *"not"* entre las que son más frecuentes. Hay que tener en cuenta, que es la que mayor frecuencia acapara dentro de los textos, lo que puede ir acompañado de otras palabras que tienen también alta frecuencia. Es decir, puede ir acompañada de palabras como *"buy, like, try, taste…"* y estas palabras en conjunción con *"not"* pueden aportar un significado negativo, pero que, de manera aislada, puede dar lugar a equívocos. Por ello es importante realizar otro tipo de pruebas para confirmar que esto es así.

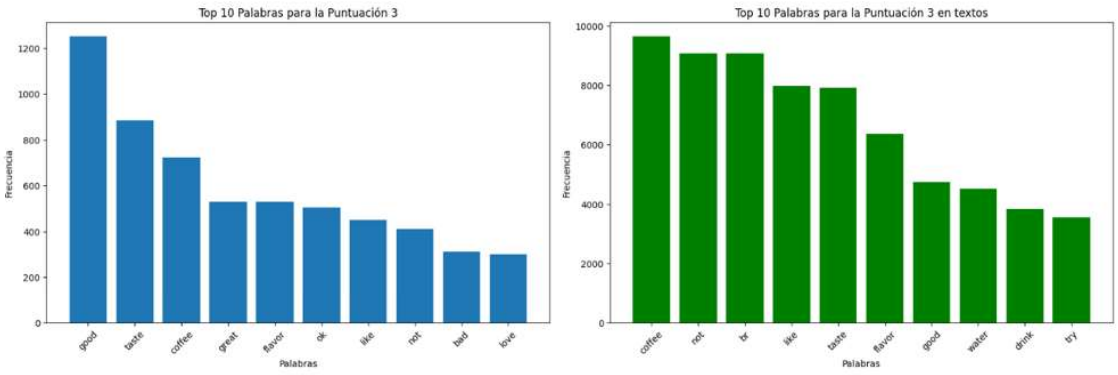


Figura 20. Mayor frecuencia de palabras para Summary y Text en función de la variable Score (Puntuación igual a 3)

En el caso de las puntuaciones neutras en el resumen como se ve en la Figura 20, es un poco más complicado de ver ya que pueden aparecer palabras que expresen aprobación y palabras que expresen disconformidad, por lo que hay que fijarse también, en la frecuencia con la que aparecen en estas reseñas. Se aprecia que palabras de buen grado son muy frecuentes (*Good, great*), pero también aparecen alguna como "*ok*" que podría indicar que el producto es correcto, que no es ni bueno, ni malo, sino que es una opinión neutra, y además aparecen, aunque con menos frecuencia, palabras de desaprobación (*not, bad*). En cómputo general sí que se podría ver como opiniones más neutras.

Para el caso de los textos completos, de nuevo se obtienen palabras que pueden otorgar críticas negativas como puede ser la aparición de la palabra "*not*", pero que de nuevo puede ir

acompañada de otras palabras como *"like, good…",* por lo que podría otorgar una crítica negativa. Pero de la misma forma, las palabras de aprobación pueden aparecer de manera aislada, de forma que otorguen una crítica correcta o positiva. Si es cierto que en este caso se ve de una manera menos clara si se otorgan un nivel de críticas balanceado, de manera que en el cómputo general sea neutro. Lo útil sería realizar un análisis de los sentimientos qué es lo que se realizará más adelante.

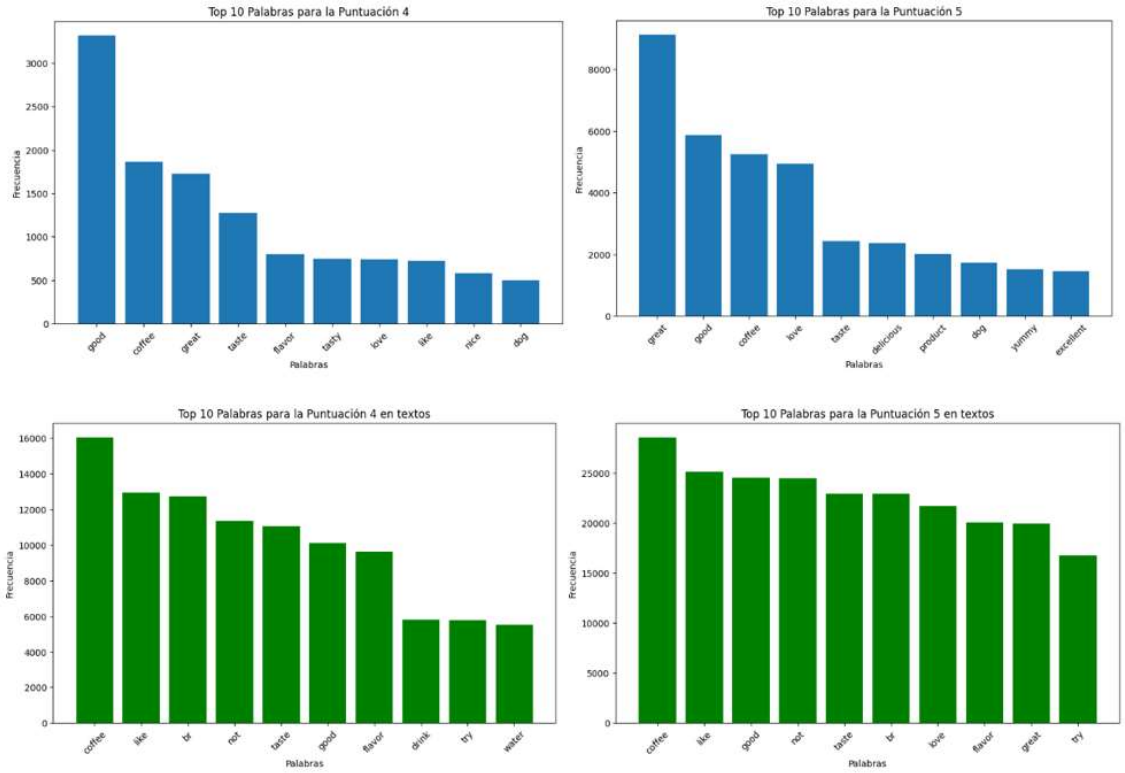


Figura 21. Mayor frecuencia de palabras para Summary y Text en función de la variable Score (Puntuación 4 y 5)

Finalmente, en el caso de las reseñas con mayores valores de puntuación como se aprecia en la Figura 21, en los resúmenes, sí que se ve claramente que solo aparecen palabras de buen grado. En este caso, no hay lugar a equívocos, ya que no aparece ninguna palabra que refleje disconformidad o desaprobación. Sí que hay distintas frecuencias de las distintas palabras que indican aprobación, y dependiendo del tipo de palabra aparece con más o menos frecuencia. En el caso de los textos, se puede apreciar como sí que aparecen con mayor frecuencia palabras de aprobación y aunque sí es verdad que aparecen palabras de desaprobación, estas aparecen con menos frecuencia que las de aprobación, por lo que en cómputo general sí que se podría interpretar que aportan aprobación. También es cierto, que es menos sencillo de ver que en el caso de los resúmenes, y esto también tiene que ver el número de muestras qué hay en los casos negativos y en los casos positivos, ya que para las reseñas de puntuación bajas existen muchas menos muestras que aquellas reseñas que tienen una buena puntuación por lo que en los casos del análisis de texto que se ha comentado anteriormente, se puede apreciar cómo si el conjunto de datos estuviera más balanceado pudiera ver mayor frecuencia de palabras negativas, pero ya se sabe que ese problema de balanceo existe y que habrá que corregir más adelante posteriormente a la culminación de estos análisis de variables.

Los resultados obtenidos en este análisis, hace pensar si hay suficiente correlación entre los textos completos y los resúmenes por lo que sería idóneo un análisis entre las dos variables los

resúmenes y el texto para ver qué tipo de correlación puede haber entre ambos. Se sabe que los resúmenes lógicamente como ya se ha analizado anteriormente, son notablemente más cortos que los textos completos, pero es útil conocer el grado de similitud que hay en cuanto a temática de unos y otros.

Por tanto, hay un tipo de análisis que van a ser muy útiles para desgranar estos interrogantes, uno es la similitud del coseno. Esta es una métrica utilizada para medir la similitud entre dos vectores en un espacio multidimensional, calculando el coseno del ángulo entre ellos. Esto es una métrica muy importante y útil en el contexto de NLP, tal como afirma Mielse P. (2023). Para aplicar la similitud de coseno en textos lo primero que se debe de hacer es representar los textos como vectores en un espacio de características, donde cada dimensión se corresponde a una palabra del vocabulario, por lo que se estará en un espacio multidimensional. Una vez vectorizados los textos, se calcula el producto de los dos vectores y se divide por el producto de sus magnitudes, lo que da un resultado entre -1 y 1, donde 1 indica que los vectores apuntan en la misma dirección, es decir, que tienen máxima similitud, 0 indica que son ortogonales por lo que no tienen similitud y -1 indican que apuntan en direcciones opuestas.

En el caso que ocupa a este trabajo, al comparar un texto completo con un resumen, esta métrica permite cuantificar cuánto de bien el resumen captura el contenido del texto original. Al representarse ambos como vectores, se puede calcular su similitud y determinar si el resumen refleja de buena manera los temas y palabras clave del texto completo. Esto es muy útil en sistemas de resumen automático y en la evaluación de coherencia entre documentos largos y sus resúmenes por lo que es idóneo para este caso.

Para su implementación, el primer paso es seleccionar los datos para el análisis que en este caso van a ser las columnas de texto y resumen ya tokenizados como se aprecia en la Figura 22. Se van a tomar todos los registros que hay en ambas columnas, por lo que en *sample* se selecciona mediante la función *shape* todas las filas que se corresponden con cada una de las dos columnas.

```
# Seleccionamos una muestra de datos para el análisis
sample_data = df_tokenized[['Transformed_Text', 'Transformed_Summary']].dropna().sample(n=df_tokenized.shape[0], replace=True, random_state=42)
```

Figura 22. Selección de datos a calcular TF-IDF

A continuación, como se muestra en la Figura 23, se realiza la vectorización TF-IDF de ambas columnas, qué permite convertir el texto en valores numéricos, para que se pueda procesar mediante la técnica de similitud de coseno. Después de realizar la normalización, la tokenización… En definitiva, la transformación de los textos que se realizó para poder hacer estos análisis univariante y bivariante. Este tendría que haber sido el último de los pasos, ya que es primordial para la aplicación de los algoritmos de aprendizaje automático. Sin embargo, es una tarea que no se ha realizado hasta este momento ya que el valor semántico de las palabras podría ser útil a la hora de realizar los análisis como así se ha podido comprobar a lo largo del presente análisis. Además, aún quedan por realizar más analisis, por lo que la matriz que se obtiene después de hacer la técnica que se explicará a continuación, se obtendrá más adelante. En este caso, se aplica esta técnica para el cálculo de la Similaridad de Coseno.

```
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(sample_data['Transformed_Text'] + sample_data['Transformed_Summary'])
```

Figura 23. Operaciones a realizar para la transformación TF-IDF

La técnica TF-IDF, se compone de TF qué es la frecuencia de término, y mide la frecuencia de la palabra en un documento en particular, en el presente caso será en los textos y en los

resúmenes. Por otro lado, se tiene IDF qué es la frecuencia inversa de documento, imite lo única que es la palabra en todo el Corpus, es decir, en la colección de documentos. En el presente caso será en todos los textos y todos los resúmenes. Por tanto, esta técnica lo que hace es combinar las dos medidas para ponderar las palabras dando peso aquellas que son frecuentes en un documento, pero raras en otros. Una de las mayores ventajas es la penalización de las palabras poco comunes, y prioriza las palabras importantes en el contexto de cada documento, en el presente caso, de cada texto y cada resumen.

Por tanto, una vez realizada la vectorización, se calcula la similitud de coseno como se puede ver en la Figura 24, mediante la función *cosine_similarity* dónde calcula la similitud del coseno entre las matrices texto y resumen devolviendo una matriz en dos dimensiones donde la fila X la columna Y contiene la similitud del texto y el resumen. Finalmente se obtiene el único valor de la matriz qué es [0][0]. Y finalmente se guarda en una lista ya inicializada para cada valor calculado en el bucle.

```
for rr, row in sample_data.iterrows():
    tfidf_text = tfidf_vectorizer.transform([row['Transformed_Text']])
    tfidf_summary = tfidf_vectorizer.transform([row['Transformed_Summary']])

    # Cosine similarity
    cosine_sim = cosine_similarity(tfidf_text, tfidf_summary)[0][0]
    cosine_similarities.append(cosine_sim)
```

Figura 24. Cálculo de la Cosine Similarity

Si se grafican los resultados mediante un gráfico de dispersión, se puede apreciar que la mayoría de los casos se tienen similitudes muy cercanas a cero, lo que hace pensar que los textos y los resúmenes no tienen un gran grado de similitud. Se puede apreciar también qué el rango en el que se puede mover la similitud es entre 0 y 0.5 de forma que a partir de ahí sí que se encuentran grados mayores de similitud, pero no llegan a ser totalmente similares. Se puede apreciar qué hay similitudes en torno a 0.4 y 0.6 qué hace pensar que se pueda captar cierto grado de similitud entre resúmenes y texto completo, sin embargo, hay una proporción significativa de textos completos y resúmenes con similitudes muy bajas qué es lo que se puede encontrar entre 0 y 0.4 aproximadamente, y esto dicta que los resúmenes no reflejan correctamente el contenido del texto completo o puede no captar la síntesis del texto completo. Por otra parte, sí que se puede encontrar entre 0.6 y 0.8, resúmenes con buena calidad ya que tiene similitudes muy altas entre el resumen y el texto completo y en este caso los resúmenes sí que captan con bastante fidelidad una síntesis del texto completo.

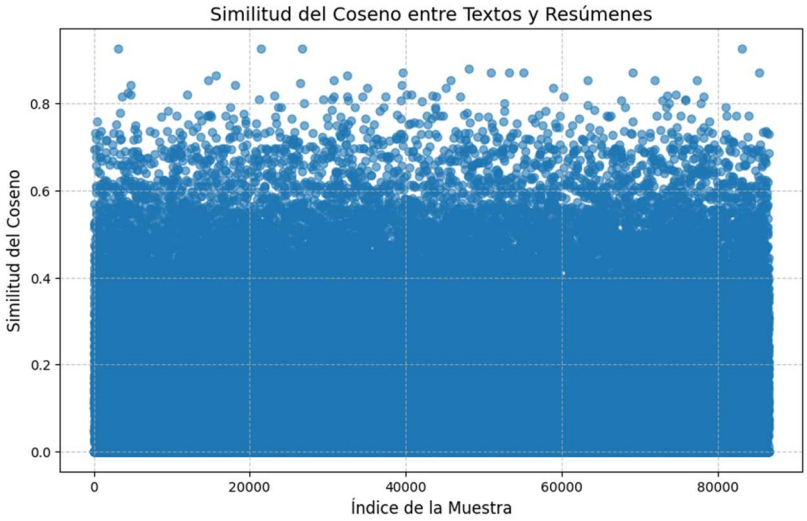


Figura 25. Gráfico de Cosine Similarity obtenido

Esta casuística, hace pensar en primer lugar que puede haber contenido irrelevante, es decir, que el resumen podría incluir información que no está presente en el texto y al revés el texto con tener información que no está sintetizada en el resumen de manera correcta. Esto es algo que analizando pequeñas muestras del dataframe se puede observar a simple vista, con lo que es algo que podría esperarse. Otra de las razones que puede dar este resultado, es el que los resúmenes podrían ser excesivamente breves en comparación con textos muy largos, y es algo que reduce notablemente la similitud de coseno. De nuevo si se realiza una inspección de los datos del dataframe, se puede observar cómo hay resúmenes de 1 o 2 palabras y luego textos con una longitud considerable en comparación con el resumen al que hacen referencia y esto podría explicar también los bajos valores obtenidos.

Esto dice mucho de la baja calidad de los resúmenes de los que se disponen el conjunto de datos, por tanto, esto hace pensar que se podría mejorar la escritura de dichos resúmenes en comparación con el texto, algo que la empresa, en este caso Amazon, podría exigir a los clientes una mejor escritura de las reseñas, para mejorar la calidad de los datos en los procesos de análisis, o bien eliminar aquellas reseñas que no se corresponden de manera fidedigna entre texto y resúmenes. Sin embargo esta eliminación, puede privar de información muy importante ya que la semántica puede jugar un papel clave a la hora de realizar un análisis de datos de este tipo de forma que se puede dar una casuística en el que un producto que dispone de 20 reseñas, puede tener después de calcular la similitud de coseno, una única reseña útil en caso de que se encuentre una similitud de coseno bastante alta y por tanto esto podría no ser útil de cara a los clientes ya que solo se dispondría de una única reseña con la que los clientes pueden comparar y podría estropear también el sistema de recomendación. Por ello, no es una buena práctica eliminar aquellas reseñas con una similitud de coseno muy baja. Tampoco es adecuado de cara a futuro, ya que se pueden eliminar datos muy importantes para el análisis del producto o productos qué forman parte de la oferta de Amazon y también evita la recolección de más datos que pueden ser útiles para análisis futuros. Una estrategia que se puede fomentar desde Amazon, es la de pequeñas recompensas para los clientes que realicen una reseña consistente entre el resumen y el texto de forma que obtengan un beneficio futuro en la compra de un producto, por ejemplo, es decir, estrategias que ayuden a conseguir una calidad de datos mejor de cara a futuro.

Por otra parte, la baja similitud entre textos y resúmenes encontrada, puede justificar la concatenación de ambas columnas para formar una única que permita utilizarse como variable objetivo para los modelos de Machine Learning con los que se puede realizar la clasificación de reseñas. Esto permite por un lado que los resúmenes aporten información complementaria que no hay en los textos, y permite también obtener una mayor riqueza semántica, así como homogeneizar los datos ya que al concatenarlos se genera un único campo qué puede ser preprocesado y analizado de manera uniforme además de que los resúmenes y los textos tienen longitudes muy distintas cómo se ha comprobado anteriormente.

Por tanto, lo que se ha demostrado es una baja correlación entre estas dos variables de resumen y textos. Esto difiere mucho del caso de qué se hubiera encontrado una alta similitud entre textos y resúmenes, y en ese caso la correlación entre ambos sería bastante elevada. Este no es el caso, pero de haber sido así con una alta correlación una de las dos columnas se podría haber eliminado ya que existirían redundancias entre ambas columnas y podría introducir ruido en el análisis. Sin embargo, en este caso no es así.

Resta saber si hay algún tipo de correlación entre los sentimientos de los resúmenes y los textos completos. Esto se puede realizar de una manera sencilla con la librería de Python Textblob. Según Auwal Khalid A. (2023), se trata de una librería de procesamiento de lenguaje natural que ayuda a realizar análisis de sentimiento calculando la polaridad del texto (positivo o negativo), así como su subjetividad (clasificando el texto en más objetivo o menos objetivo), entre otro tipo de tareas cómo traducción de idiomas, detección de idiomas…

En el caso que ocupa este proyecto, se debe de tener en cuenta algunos detalles. El primero es la polaridad, esto refleja la puntuación de sentimiento y puede variar entre -1 siendo muy negativo o 1 como muy positivo, siendo el cero el que indica neutralidad.

```
# Calculamos el sentimiento para ambas columnas
df_tokenized['Sentiment_Text'] = df_tokenized['Transformed_Text'].apply(lambda text: TextBlob(str(text)).sentiment.polarity)
df_tokenized['Sentiment_Summary'] = df_tokenized['Transformed_Summary'].apply(lambda text: TextBlob(str(text)).sentiment.polarity)

# Calculamos la correlación entre los sentimientos
correlation = df_tokenized['Sentiment_Text'].corr(df_tokenized['Sentiment_Summary'])

# Mostramos los resultados
print("Correlación entre los sentimientos de texto y resumen:" + str(correlation))
```

```
Correlación entre los sentimientos de texto y resumen:0.2725784021783495
```

Figura 26. Cálculo de la correlación entre sentimientos de Resumen y Texto

Para realizar este propósito se define una función lambda para aplicarlo a todas las filas en el que se toma el texto de cada una de las filas para las columnas que contienen los textos y los resúmenes transformados (*Transformed_Text* y *Transformed_Summary*), y sobre cada texto se convierte a objeto *TextBlob* y a continuación se calcula la polaridad del sentimiento utilizando la propiedad *.sentiment.polarity* como se puede apreciar en la Figura 26. La correlación entre las puntuaciones de sentimiento de ambas columnas da como resultado 0.27. Da una relación positiva pero muy pequeña ya que cuando el sentimiento del texto aumenta tienda aumentar ligeramente el sentimiento del resumen sin embargo no es algo lo suficientemente consistente como para poder afirmar que efectivamente es así ya que se necesitaría un grado de correlación más alto para que fuese lo suficientemente positivo.

Esto puede indicar discrepancias en las polaridades de sentimiento, pero viene de una situación en la que los resúmenes son notablemente más cortos que las reseñas en sí con lo que se tiende a condensar mucha información qué hay en los textos completos.

Antes de iniciar la siguiente fase que ya va a ser la de empezar a transformar el dataframe para que pueda ser tratado por los modelos de Machine Learning, se eliminarán todas aquellas columnas que se han creado durante el análisis univariante, bivariante… para dejar únicamente cuatro columnas, que serán las de identificador de producto, puntuación, texto completo de la reseña y resumen tokenizados.

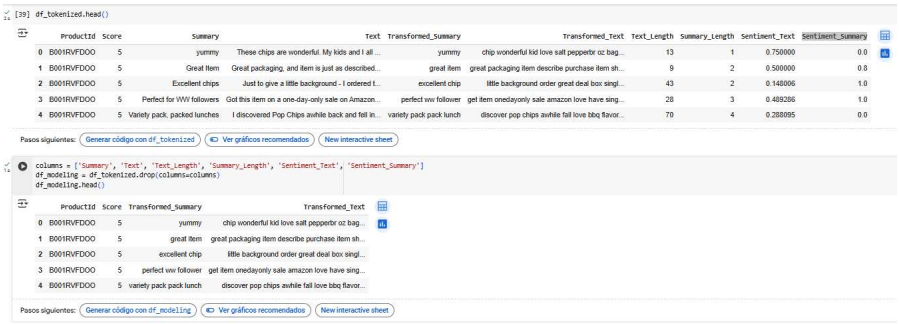


Figura 27. Eliminación de columnas irrelevantes para el análisis

El análisis multivariante no es necesario en este caso, ya que el objetivo principal es clasificar reseñas en función de las puntuaciones (*Score)* usando los textos transformados. Las relaciones clave entre las variables han sido exploradas en el análisis univariado y bivariado, identificando que las columnas Transformed_Text y Transformed_Summary aportan información complementaria y que la puntuación está sesgada hacia valores altos. Además, los datos textuales ya han sido transformados y normalizados, siendo suficientes para representar el problema y entrenar el modelo de clasificación.

Detección de nulos

Aplicando la detección de nulos, como se aprecia en la Figura 28, a todas las columnas del data frame se puede apreciar cómo en la única que aparecen valores faltantes es en la del resumen, concretamente 376 resumenes vacíos. En este caso no va a haber necesidad de hacer un tratamiento de estos valores faltantes ya que se va a realizar una concatenación de las columnas de texto y resumen, para obtener la variable objetivo que serían los textos.

```
Valores faltantes:

ProductId: 0
Score: 0
Transformed_Text: 0
Transformed_Summary: 376
```

Figura 28. Resumen de nulos detectados

Otra cosa es sí hubieran valores faltantes en las variables identificador de producto o puntuación, en ese caso la forma de proceder hubiera sido distinta, dependiendo del número de valores faltantes. Pero en este caso al concatenar posteriormente estas dos columnas no va a hacer falta realizar un tratamiento especial, ya que la falta de un resumen en una reseña concreta a la hora de realizar la concatenación unicamente en el resultado se obtendrá lo que hubiera en la variable del texto completo de la reseña. Sin embargo, si hay que tener cuidado al concatenar, ya que los valores nulos en la concatenación, deberán ser sustituidos por cadenas vacías para evitar problemas en el proceso.

Valores atípicos

Para el caso de los valores atípicos de las variables cuantitativas (*Score*), se puede apreciar en la Figura 29, como hay valores atípicos que se corresponden con aquella reseñas que tienen una puntuación muy baja de 1 o 2. En este caso si se eliminan estas reseñas, se estarían perdiendo las reseñas de de muy baja puntuación y baja puntuación lo cual sería malo para el entrenamiento del modelo, por lo que aunque lo ortodoxo sería eliminar los valores atípicos, en este caso como se ha estudiado previamente en los análisis previos no es recomendable, ya que, se estaría perdiendo información semántica muy valiosa para los modelos de aprendizaje. Este gráfico es lógico debido a la distribución de las reseñas por puntuación y era algo totalmente esperado.

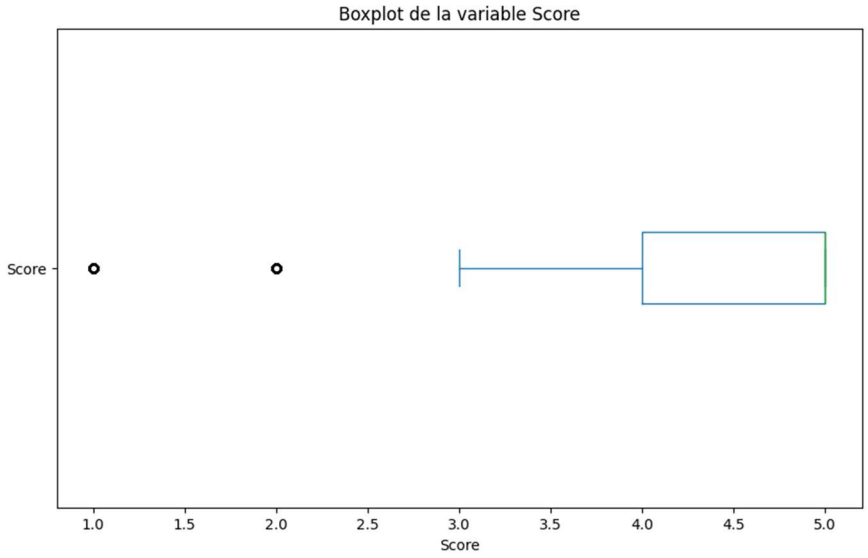


Figura 29. Boxplot de outliers detectados

Concatenación de columnas de Texto y Resumen

A continuación se concatenan las columnas de texto y resumen ya tokenizados, como se puede ver en la Figura 30, en una única columna ya que esto permitirá una mejor integración en los modelos de Machine Learning y por ende un análisis más sencillo de los datos. Para ello es importante utilizar la función fillna the Python en la contratación ya que cómo sea comprobado anteriormente existen valores nulos en el caso de la columna de resúmenes, por tanto para evitar problemas en las concatenación se debe de rellenar esos valores y en este caso se hará con una cadena vacía por lo que finalmente solo quedarán en los casos en los que no había resumen unicamente el contenido de la columna de texto tokenizado.

```
df_modeling_patch['Review'] = df_modeling_patch['Transformed_Summary'].fillna('') + " " + df_modeling_patch['Transformed_Text'].fillna('')
df_modeling_patch.head()
```

	ProductId	Score	Transformed_Summary	Transformed_Text	Review
0	B001RVFDOO	5	yummy	chip wonderful kid love salt pepperbr oz bag...	yummy chip wonderful kid love salt pepperbr ...
1	B001RVFDOO	5	great item	great packaging item describe purchase item sh...	great item great packaging item describe purch...
2	B001RVFDOO	5	excellent chip	little background order great deal box singl...	excellent chip little background order great...
3	B001RVFDOO	5	perfect ww follower	get item onedayonly sale amazon love have sing...	perfect ww follower get item onedayonly sale a...
4	B001RVFDOO	5	variety pack pack lunch	discover pop chips awhile fall love bbq flavor...	variety pack pack lunch discover pop chips awh...

os siguientes: Generar código con df_modeling_patch | Ver gráficos recomendados | New interactive sheet

```
# Eliminamos las columnas de Texto y Resumen, una vez estan ya concatenadas
df_modeling_patch.drop(['Transformed_Text', 'Transformed_Summary'], axis=1, inplace=True)
df_modeling_patch.head()
```

	ProductId	Score	Review
0	B001RVFDOO	5	yummy chip wonderful kid love salt pepperbr ...
1	B001RVFDOO	5	great item great packaging item describe purch...
2	B001RVFDOO	5	excellent chip little background order great...
3	B001RVFDOO	5	perfect ww follower get item onedayonly sale a...
4	B001RVFDOO	5	variety pack pack lunch discover pop chips awh...

Figura 30. Concatenación de columnas de texto y resumen tokenizadas

Posteriormente las columnas de texto y resumen que se han venido utilizando hasta este punto, se eliminan ya que no serán necesarias para las siguientes fases de modelado ya que se proseguirá con la columna *Review*.

Creación de una feature para agrupar reseñas por tipos de puntuación

A continuación, como se ha comentado en puntos anteriores, se va a proceder a agrupar las reseñas en positivas, negativas o neutras, como se indica en la Figura 31. Esto se hace ya que se dispone de 5 valores de puntuación desde 1 a 5 pero se requiere para la clasificación de reseñas en positivas o negativas realizar una agrupación, que permita realizar un mejor entrenamiento de los datos en la fase de modelado. Por tanto, se agruparán las reseñas con valores 1 o 2 como reseñas negativas, las reseñas con valor 3 serán las reseñas neutras, y finalmente las reseñas con valores 4 o 5 serán reseñas positivas.

```
# Creamos la nueva feature a partir de Score para clasificar en buenas o malas las reseñas.
df_modeling_patch['Label'] = df_modeling_patch['Score'].apply(lambda x: 'negative' if x in [1, 2] else ('neutral' if x == 3 else 'positive'))

# Reordenamos las columnas para que la nueva columna aparezca antes de 'Score'
columns = list(df_modeling_patch.columns)
columns.remove('Label')
columns.insert(columns.index('Score'), 'Label')
df_modeling_patch = df_modeling_patch[columns]
df_modeling_patch.head(40)
```

	ProductId	Label	Score	Review
0	B001RVFDOO	positive	5	yummy chip wonderful kid love salt pepperbr ...
1	B001RVFDOO	positive	5	great item great packaging item describe purch...
2	B001RVFDOO	positive	5	excellent chip little background order great...
3	B001RVFDOO	positive	5	perfect ww follower get item onedayonly sale a...
4	B001RVFDOO	positive	5	variety pack pack lunch discover pop chips awh...
5	B001RVFDOO	positive	4	love flavor salt pepper like chip entire famil...
6	B001RVFDOO	neutral	3	chip popchip want try bake barbeque one okay r...
7	B001RVFDOO	positive	4	good sampler pack flavor like pop chips lot fl...

Figura 31. Operaciones para la creación de la Feature a partir de Score

Esta nueva *feature,* se llamará *Label* y se obtiene mediante una función Lambda aplicada sobre la columna de puntuaciones de las reseñas (*Score*) de forma que el valor que toma esta *feature* viene dado o etiquetado por el razonamiento realizado anteriormente, dónde es negativa en caso de que el valor de la puntuación sea 1 o 2, neutra en caso de que sea 3 y positiva en caso de que sea 4 o 5 el valor de la puntuación. Posteriormente la columna de puntuación se elimina

ya que no será necesaria en los procesos posteriores, para el modelado y entrenamiento con Machine Learning.

En este punto, como se muestra en la Figura 32, se verifica la distribución de la variable *Score* para comprobar de nuevo la cantidad de reseñas que se tienen para cada valor de puntuación y posteriormente, se elimina la columna *Score*, ya que no será de utilidad para los análisis siguientes.

```
print("Distribución antes de SMOTE:")
print(df_modeling_patch['Score'].value_counts())
```

```
Distribución antes de SMOTE:
Score
5    51435
4    15347
3     8754
1     6103
2     4926
Name: count, dtype: int64
```

Figura 32. Distribución numérica de las valoraciones de reseñas del conjunto de datos

Cómo se ha comentado anteriormente, se aprecia que las clases correspondientes a una valoración positiva son muy dominantes frente a las clases qué muestran valoraciones neutras o negativas.

Otro aspecto importante, y qué es crucial a partir de este punto del proyecto, es la de ir almacenando las diferentes transformaciones que se van realizando a lo largo del proyecto, en ficheros CSV. Esto tiene una explicación, y no es otra que la de ahorrar memoria RAM del entorno de ejecución ya que los recursos son muy limitados. Por tanto, no es de extrañar que, en el código, aparezcan guardados de los distintos dataframes sobre los que se ha ido realizando transformaciones, y posteriores lecturas de ficheros CSV para poder cargar a partir de ese punto datos ya transformados sin tener que ejecutar todas las celdas desde el principio del fichero. ipynb.

Esto permite como se ha comentado, ahorrar memoria RAM, algo que es crucial sobre todo a partir de este punto, donde se van a producir transformaciones muy costosas computacionalmente como se explicará a continuación, y además permite ahorrar recursos ya que a la hora de retomar el proyecto se puede realizar en un punto avanzado sin tener que realizar todas las transformaciones desde el conjunto original de 568000 filas.

Codificación de la variable o columna del dataframe ‘Label’

Se realiza la carga del dataframe después de haber realizado la *Feature*, cómo se puede apreciar en la Figura 33, y se realizan a continuación las operaciones necesarias para empezar a realizar el balanceo de las clases qué hay en el conjunto de datos en función de la variable *Label*, de forma que posteriormente en el modelado se produzca una correcta detección de las diferentes clases involucradas.

```
[ ] # Cargamos el csv_modelado
    df_SMOTE = pd.read_csv('/content/clean_reviews.csv')
    df_SMOTE.head(10)
```

	ProductId	Label	Review
0	B001RVFDOO	positive	yummy chip wonderful kid love salt pepperbr ...
1	B001RVFDOO	positive	great item great packaging item describe purch...
2	B001RVFDOO	positive	excellent chip little background order great...
3	B001RVFDOO	positive	perfect ww follower get item onedayonly sale a...
4	B001RVFDOO	positive	variety pack pack lunch discover pop chips awh...

Figura 33. Carga del dataframe para realizar el balanceo de datos

Antes de proceder con el balanceo de datos, es importante realizar la codificación de la variable *Label*. Dado que, en el análisis de modelado posterior, se va a hacer uso de la Regresión Logística y se puede apreciar cómo la variable *Label* tiene un orden asociado, además de qué se trata de la variable objetivo y se procederá a resolver un problema de clasificación, se puede aplicar *LabelEncoder.* Estas premisas son importantes ya que si no se debería utilizar otro codificador. El resultado es el que se puede apreciar en la Figura 34. Destacar que la variable codificada se llamará *Label_Encoded*, por lo que la variable *Label* no será necesaria de ahora en adelante y se podrá eliminar. Ahora, se procederá a realizar las operaciones necesarias para realizar el balanceo de los datos.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Codificamos la columna Label con LabelEncoder para la variable objetivo
label_encoder = LabelEncoder()
df_SMOTE['Label_Encoded'] = label_encoder.fit_transform(df_SMOTE['Label'])

# Eliminamos la columna original Label, ya que no se necesitará más adelante
df_SMOTE.drop(columns=['Label'], inplace=True)

print("Primeras filas del DataFrame con Label codificado:")
df_SMOTE.head()
```

```
Primeras filas del DataFrame con Label codificado:
    ProductId                                             Review  \
0  B001RVFDOO  yummy chip wonderful kid love salt pepperbr   ...
1  B001RVFDOO  great item great packaging item describe purch...
2  B001RVFDOO  excellent chip little background   order great...
3  B001RVFDOO  perfect ww follower get item onedayonly sale a...
4  B001RVFDOO  variety pack pack lunch discover pop chips awh...

   Label_Encoded
0              2
1              2
2              2
3              2
4              2
```

Figura 34. Codificación de la variable Label

Balanceo de datos

Para el balanceo de datos, hay diversas estrategias. La primera de ellas es aumentar las muestreas de las clases minoritarias para que tenga la misma cantidad de datos que las clases mayoritarias. Esto es aumentar el número de muestras que tengan un valor de *Label_Encoded*, neutral o negative. El resultado es un conjunto de datos que llega a las 190000 filas, lo que engorda notablemente el conjunto de datos del que se parte de alrededor de 86000 filas. Idealmente sería la mejor opción, ya que se mantiene toda la semántica de las reseñas de las clases mayoritarias (reseñas etiquetadas como positivas). Sin embargo, realizando pruebas no se logró realizar la transformación de los datos posteriormente. Esto es, porque se trata de un volumen de datos demasiado grande para que *Colab* en su versión gratuita pueda manejarlo, ya que la cantidad de memoria RAM de la que se dispone es muy limitada para este volumen de datos.

Esto hace que se empiece a pensar en otras estrategias. Una de ellas es equiparar mediante submuestreo, las muestras de las que se dispone en la clase mayoritaria (*positive*) a una cantidad

de datos similar a la que se tienen en las clases minoritarias. En la Figura 35, se puede apreciar cómo aplicando esta estrategia hace que el volumen de datos se reduzca considerablemente, sin embargo, esto también hace que se pierda una grandísima cantidad de información y muy válida, ya que las muestras originales de las reseñas etiquetadas como positivas se reducen de manera considerable. Esto hace que después en la transformación de datos, el entorno lo pueda manejar con la cantidad de memoria RAM de la que se dispone, pero semánticamente se estaría perdiendo una cantidad de información muy valiosa para el análisis, por lo que no es del todo óptimo utilizar esta estrategia.

```
Distribución antes de SMOTE:
Label
positive    66782
negative    11029
neutral      8754
Name: count, dtype: int64
```

Figura 35. Distribución de clases en Label_Encoded

Finalmente, se encuentra una estrategia intermedia. Esta se trata de, por un lado, aumentar el número de muestras de las clases minoritarias (*neutral* y *negative*), así como reducir en un porcentaje mediante submuestreo las clases de las que se tienen más muestras. Este es un enfoque intermedio, y es el más razonable a tenor de las circunstancias ya que se preserva la mayor cantidad posible de información original de las clases mayoritarias, y por otra parte se generan la menor información posible de las clases minoritarias, ya que estás clases la información que se va a obtener es sintética. Cierto es, que para las clases minoritarias se utiliza la técnica SMOTE, lo que hace que la calidad de los datos sea buena, o al menos mejor que simplemente replicar muestras de las clases minoritarias.

Por tanto, el primer paso será submuestrear la clase positiva y reducir su número de muestras. Se hace uso de RandomUnderSampler partiendo de lo comentado por Ersan Yagci H. (2021). Esta técnica reduce el tamaño de la clase mayoritaria en el conjunto de datos para equilibrarlo con la clase minoritaria. Realiza un muestreo aleatorio, y selecciona un conjunto de datos de la clase mayoritaria hasta alcanzar un nivel de equilibrio con la clase minoritaria o hasta el nivel que se ha especificado. En este caso, se especifica un nivel al cual se desea muestrear la clase mayoritaria.

```python
print("Distribución de clases antes del submuestreo:")
print(df_SMOTE['Label_Encoded'].value_counts())

undersampler = RandomUnderSampler(sampling_strategy={2: 30000}, random_state=42)  # Reducir clase 2 a 30,000 muestras

X_text = df_SMOTE['Review'].fillna('')  # Columna de texto
y_labels = df_SMOTE['Label_Encoded']    # Etiquetas
product_ids = df_SMOTE['ProductId']     # Columna ProductId

# Creamos un DataFrame combinado para el submuestreo
df_combined = pd.DataFrame({
    'Review': X_text,
    'Label_Encoded': y_labels,
    'ProductId': product_ids
})

# Submuestreo
df_under = pd.DataFrame(undersampler.fit_resample(df_combined, df_combined['Label_Encoded'])[0])

# Recuperamos las columnas submuestreadas
X_under_text = df_under['Review']         # Textos submuestreados
y_under_labels = df_under['Label_Encoded']  # Etiquetas submuestreadas
product_ids_under = df_under['ProductId']  # ProductId submuestreado

# Verificación del submuestreo y mostramos una muestra
print("Distribución de clases después del submuestreo:")
print(y_under_labels.value_counts())
df_under.head()
```

Figura 36. Proceso de submuestreo con RandomUnderSampler

En la figura 36, se puede apreciar el proceso de submuestreo mediante esta técnica. En esencia se toma el argumento *sampling_strategy*, y se le especifica la clase de la cual se quiere obtener el número de muestras determinado, en este caso de la clase dos se desean obtener 30000 muestras, lo cual va a reducir de 64000 muestras aproximadamente de la clase positiva, a 30000. Es cierto, qué es un submuestreo muy elevado, pero es el máximo número de que, junto al aumento de muestras de las otras dos clases minoritarias, se consigue que el entorno funcione correctamente sin limitaciones de memoria RAM. Posteriormente, se obtienen las distintas columnas del dataframe original, se combinan en un nuevo dataframe *df_combined*, para facilitar el proceso de submuestreo, se realiza el submuestreo y posteriormente se recuperan las columnas muestreadas.

Con ello ya se tiene el dataframe *df_under* submuestreado. En la figura 37 se puede apreciar ahora, la distribución de muestras en función de cada clase.

```
Distribución de clases después del submuestreo:
Label_Encoded
2    30000
0    11029
1     8754
Name: count, dtype: int64
```

Figura 37. Distribución de clases después del submuestreo

En este punto, cabe destacar la utilización del comando '*%whos'*, que permite ejecutado en una celda, observar el número de variables cargadas en memoria. Con ello, se eliminan mediante el comando '*del*' algunos dataframes que ya no son de utilidad ya que han sido utilizados para transformaciones posteriores dejando únicamente aquel dataframe que sirve para las operaciones posteriores (*df_under*).

A continuación, se procede a sobre muestrear las clases minoritarias mediante la técnica SMOTE. Como se ha comentado anteriormente, esta técnica genera muestras sintéticas a partir de las muestras ya existentes. Según Brownlee J. (2021), Esto se realiza mediante interpolación, donde en una primera fase se realiza una identificación de ejemplos o muestras de la clase minoritaria, posteriormente se aplica la técnica de detección de los k vecinos más cercanos de las muestras seleccionadas utilizando alguna métrica como la distancia euclídea. Finalmente elige aleatoriamente 1 de los vecinos más cercanos. Todo esto se realiza tantas veces como sea necesario hasta conseguir el número de muestras especificado o deseado.

Para poder aplicar esta técnica, es necesario poder tener características numéricas en todo el conjunto de datos, es por ello que se requiere codificar la columna de identificador de producto. En este caso, no se ve claramente un orden establecido de productos, es decir, no hay una clasificación clara donde se vean qué producto va detrás de qué otro producto. Si se recuerda del caso de la columna *Score*, en ese caso sí había un orden ya que era numérico y además era incremental de 1 a 5, pero en este caso son identificadores de los cuales no se sabe cuál va delante de otro. Por tanto, no se puede hacer uso de *LabelEncoder*. De forma que se codifica mediante *One-Hot Encoding*, que como mayor contrapartida tiene la gran dimensionalidad que se genera al utilizar esta técnica. Esto ya hace pensar que posteriormente habría que aplicar técnicas de reducción de la dimensionalidad, pero se verá más adelante. Con esta técnica, cada producto se representa como un vector binario donde solo una posición tiene un valor 1, donde el resto son ceros.

El siguiente paso para poder realizar la técnica SMOTE, es convertir el texto a una representación numérica, y esto se realiza mediante la generación TF-IDF. Esta técnica ya fue descrita con anterioridad en los análisis de las variables, concretamente en el análisis bivariante para el

cálculo de la *Cosine Similarity*. La única salvedad, es que en este caso al contrario que para el cálculo de la característica Cosine Similarity, se debe de ajustar el límite de características para evitar que se cree un conjunto de datos excesivamente grande. Esta es una de las limitaciones que se crean en el entorno gratuito, ya que posteriormente en el manejo de la RAM se pueden generar hasta 42000 columnas. Por tanto, se debe limitar el número de características para que el conjunto de datos transformado no sea excesivamente grande, y no desborde la memoria RAM del entorno. Se puede apreciar esto en la Figura 38.

```
# Generamos TF-IDF con un límite de características para evitar que se cree un dataset demasiado grande
tfidf = TfidfVectorizer(max_features=6000)  # Límite del vocabulario a las 6000 palabras más frecuentes
X_tfidf_under = tfidf.fit_transform(df_under['Review'].fillna(''))
```

Figura 38. TF-IDF con límite de vocabulario

Posteriormente, se puede apreciar en la Figura 39, cómo se aplica la técnica SMOTE por bloques, debido a que si no se aplica por bloques da problemas de desborde de la memoria RAM. Por tanto, se debe crear un flujo para que se procese por bloques, ya que en este punto la dimensionalidad obtenida de la transformación TF-IDF es de unas 49000 filas por unas 6000 columnas aproximadamente, lo que hace un conjunto de datos excesivamente elevado para la cantidad de RAM limitada de la que se dispone. Por tanto, se selecciona un tamaño de lote de 10000, y a continuación se especifica la técnica SMOTE a aplicar, para ampliar las clases minoritarias al mismo número de la clase mayoritaria. Es importante también, especificar el número de muestras de la clase mayoritaria que se obtuvo a partir del submuestreo, qué son 30000 muestras. Esto es así porque si no no funcionará adecuadamente.

```
# Balanceo las clases
batch_size = 10000  # Tamaño de los bloques para el procesamiento por lotes
smote = SMOTE(sampling_strategy={0: 30000, 1: 30000, 2: 30000}, random_state=42)

X_resampled_list = []
y_resampled_list = []

for i in range(0, X_combined_under.shape[0], batch_size):
    # Selección de un bloque de datos
    X_batch = X_combined_under[i:i+batch_size]
    y_batch = y_under.iloc[i:i+batch_size]

    # Verificación de que el bloque contiene al menos dos clases
    if len(y_batch.unique()) > 1:
        # Aplicación de SMOTE al bloque
        X_resampled_batch, y_resampled_batch = smote.fit_resample(X_batch, y_batch)

        # Guardamos los resultados del bloque
        X_resampled_list.append(X_resampled_batch)
        y_resampled_list.extend(y_resampled_batch)
    else:
        print(f"Bloque omitido en el rango {i}:{i+batch_size} debido a una única clase.")

# Combinación de los bloques resultantes
X_resampled = vstack(X_resampled_list)
y_resampled = pd.Series(y_resampled_list)

print("SMOTE aplicado. Nuevas dimensiones:", X_resampled.shape)
```

Figura 39. Aplicación de SMOTE por lotes

Este bucle, procesa el conjunto de datos en bloques de 10000 filas para evitar estos problemas de memoria. Se aplica SMOTE para equilibrar las clases generando ejemplos sintéticos hasta llegar a las 30000 muestras por cada clase. Este bloque también ignora todos aquellos bloques que contienen solamente una clase ya que la técnica SMOTE necesita al menos 2 clases para funcionar, si no daría error en el procesamiento. Los bloques qué han sido ampliados en muestras se van guardando en las listas *X_resampled_list* y *Y_resampled_list*. Posteriormente, se combinan los bloques procesados almacenados en estas listas en una única matriz (*X_resampled*) y una serie de etiquetas (*y_resampled*). Este es el punto más interesante de este procesamiento.

Finalmente, después de aplicar esa técnica para aumentar el número de muestras de las clases minoritarias, se debe de decodificar el identificador de producto, ya que este va a ser utilizado después en la división de conjuntos de entrenamiento y de test, por lo que el actualmente está codificado como variable numérica, y se recupera ahora para obtener el identificador como corresponde. Una vez realizada esta operación se puede observar cómo el resultado es el esperado en la Figura 40. Se observa, cómo el texto ha sido vectorizado adecuadamente donde después de aplicar la técnica SMOTE, el conjunto de datos tiene unas dimensiones de 90000 filas y 6250 columnas. En ese punto no se había realizado la decodificación de la variable *ProductId* por lo que después de la de codificación de dicha variable, se consiguen unas dimensiones de 90000 filas y 6002 columnas, dónde 6000 se corresponden con las características del texto vectorizadas, y las dos columnas restantes se corresponden con las variables *ProductId* y *Label_Encoded*. También se puede apreciar, la distribución de las etiquetas de las que se dispone después de aplicar la técnica SMOTE y ahora se tienen 30000 muestras de cada clase (positiva, negativa y neutra). Ya se dispone del conjunto de datos balanceado.

```
Dimensiones TF-IDF + ProductId (después del submuestreo): (49783, 6250)
Bloque omitido en el rango 0:10000 debido a una única clase.
Bloque omitido en el rango 20000:30000 debido a una única clase.
Bloque omitido en el rango 30000:40000 debido a una única clase.
Bloque omitido en el rango 40000:50000 debido a una única clase.
SMOTE aplicado. Nuevas dimensiones: (90000, 6250)
   abandon  ability  able  aboutbr  abovebr  abr  absence  absent  absolute  \
0      0.0      0.0   0.0      0.0      0.0  0.0      0.0     0.0       0.0
1      0.0      0.0   0.0      0.0      0.0  0.0      0.0     0.0       0.0
2      0.0      0.0   0.0      0.0      0.0  0.0      0.0     0.0       0.0
3      0.0      0.0   0.0      0.0      0.0  0.0      0.0     0.0       0.0
4      0.0      0.0   0.0      0.0      0.0  0.0      0.0     0.0       0.0

   absolutely  ...  ziploc  ziplock  zipper  zombie  zone  zuke  zukes  \
0         0.0  ...     0.0      0.0     0.0     0.0   0.0   0.0    0.0
1         0.0  ...     0.0      0.0     0.0     0.0   0.0   0.0    0.0
2         0.0  ...     0.0      0.0     0.0     0.0   0.0   0.0    0.0
3         0.0  ...     0.0      0.0     0.0     0.0   0.0   0.0    0.0
4         0.0  ...     0.0      0.0     0.0     0.0   0.0   0.0    0.0

   zukesbr  Label_Encoded   ProductId
0      0.0              0  B006N3IE6A
1      0.0              0  B004FGWU9O
2      0.0              0  B004FGWU9O
3      0.0              0  B004FGWU9O
4      0.0              0  B004FGWU9O

[5 rows x 6002 columns]
Distribución de clases después de SMOTE:
Label_Encoded
0    30000
1    30000
2    30000
Name: count, dtype: int64
```

Figura 40. Resultado del proceso de aumento de muestras con SMOTE

En este punto de nuevo, se hace uso del comando '*%whos*' para buscar variables que han sido utilizadas de manera intermedia en las transformaciones para aplicar estos muestreos, y poder eliminar dichas variables intermedias para ahorrar memoria RAM. Esto es importante ya que el flujo de trabajo se reinició en numerosas ocasiones debido a que no se hacía un uso eficiente de la RAM, por tanto, se localizan variables intermedias que ya no son de utilidad, se eliminan y se libera espacio de memoria RAM.

Reducción de la dimensionalidad

En este punto, se puede apreciar qué hay un conjunto de datos excesivamente grande para las posibilidades de la memoria RAM del entorno de trabajo. Por tanto, se hace necesario aplicar técnicas de reducción de la dimensionalidad del dataframe del que se dispone. Para ello, Verma Y. (2021), indica una forma de hacer la reducción de la dimensionalidad, con la técnica Truncated SVD. Esta técnica funciona muy bien con datos dispersos, esto es con datos con muchos valores cero, y es justamente el caso del que se dispone en este punto, ya que, si se toma conciencia de los datos de la figura 40, se puede observar cómo hay muchos valores cero en ese conjunto de

datos. Se trata de una técnica similar a PCA, y es útil en grandes conjuntos de datos donde la dimensionalidad alta puede ser un problema, cómo es este caso. Por tanto, dada la naturaleza del problema, y los tipos de datos implicados, se puede hacer uso de esta técnica.

Cómo se puede apreciar en la Figura 41, en primer lugar, se separan las columnas a las que no se van a someter esta reducción de la dimensión calidad, qué es el identificador de producto y la puntuación codificada. A continuación, se declara el número de componentes máximo al que se puede reducir la dimensionalidad del conjunto de datos. Este es el mayor número de datos posible qué se ha podido conseguir sin que hubiera problemas de memoria RAM en el procesamiento, pero se verá más adelante que no ha sido un problema para el desarrollo del estudio. Por tanto, se parte de un conjunto de datos de 90000 filas y 6002 columnas, y posteriormente de aplicar esta técnica se consigue un conjunto de datos de 90000 filas y 102 columnas, de las cuales 100 se corresponde a la reducción de la dimensión calidad del texto vectorizado, y las otras dos se corresponden con las variables que se han separado previamente para que no se sometan a esta reducción de la dimensionalidad.

```
# Separamos las columnas que no se transformarán
columns_to_exclude = ['Label_Encoded', 'ProductId']
X = df_resampled.drop(columns=columns_to_exclude)  # Características para reducción de dimensionalidad

# Inicializamos TruncatedSVD a un número máximo de 100 componentes
n_components = 100
svd = TruncatedSVD(n_components=n_components, random_state=42)
```

Figura 41. Inicialización de Truncated SVD

Destacar también, que se ha aplicado de nuevo procesamiento por lotes (batches), para que la sesión no fuese cerrada por desbordamiento de la memoria RAM. Por lo tanto, el procesamiento se va realizando por bloques, se reunifica cada bloque en un dataframe y posteriormente se añade a dicho dataframe las columnas que han sido separadas previamente a la realización de este proceso. El resultado se puede apreciar en la figura 42 dónde se obtiene la dimensión del dataframe reducido, así como las columnas reducidas desde la componente 1 a la componente 100, más las dos columnas correspondientes al identificador de producto y a la valoración codificada.

```
Dimensiones del DataFrame reducido: (90000, 102)
   Label_Encoded   ProductId  Component_1  Component_2  Component_3  \
0              0  B006N3IE6A     0.224868     0.169732     0.001875
1              0  B004FGWU9O     0.335392     0.332558     0.032498
2              0  B004FGWU9O     0.248789     0.057696    -0.000094
3              0  B004FGWU9O     0.175345     0.118058     0.001378
4              0  B004FGWU9O     0.163916     0.045063     0.003920

   Component_4  Component_5  Component_6  Component_7  Component_8  ...  \
0    -0.106909    -0.104401    -0.016099    -0.101180    -0.044705  ...
1     0.126668     0.185692     0.026451     0.139494    -0.056488  ...
2    -0.048604    -0.066944    -0.006936    -0.044778    -0.025510  ...
3     0.038473     0.080063    -0.011479     0.031160    -0.029120  ...
4    -0.034574    -0.040336     0.006436     0.002659    -0.006013  ...

   Component_91  Component_92  Component_93  Component_94  Component_95  \
0      0.023525      0.023815      0.029896      0.002447     -0.020722
1     -0.047494     -0.006288     -0.009411     -0.002372     -0.002309
2     -0.048774     -0.049600     -0.023865     -0.042802      0.043735
3     -0.016056      0.014435      0.030339      0.024132      0.028569
4      0.010476     -0.031530      0.009616     -0.022239     -0.015841

   Component_96  Component_97  Component_98  Component_99  Component_100
0     -0.028937      0.003256     -0.009755      0.025572       0.014154
1      0.021895      0.003238     -0.017743      0.019961       0.012333
2      0.002682      0.020963      0.012858      0.014631       0.002582
3     -0.012679     -0.056665     -0.010001      0.045468       0.011150
4      0.001402      0.008982     -0.001204      0.017683       0.020247

[5 rows x 102 columns]
```

Figura 42. Forma y dimensiones del dataframe reducido final

Finalmente, el resultado de este dataframe reducido se guarda en un fichero CSV, para poder partir de aquí en los análisis posteriores sin tener que empezar desde el principio realizar todas las operaciones, lo que de nuevo ahorrará gran cantidad de memoria RAM y si no se realiza de esta manera no se puede llegar a realizar los análisis posteriores debido a las limitaciones del entorno gratuito.

Escalado de las variables

En este punto, se realiza la carga de los datos a partir del fichero CSV obtenido del punto anterior, y se procede al escalado de los datos para su posterior procesamiento y modelado. Este es un punto importante, ya que permite una mejora del rendimiento del modelo cómo pueden ser modelos de redes neuronales o algoritmos que sean sensibles a la escala de los datos ya que mejora la convergencia durante el entrenamiento y la precisión del modelo. También evita sesgos por magnitudes, ya que si una característica tiene un rango mucho mayor que otra puede introducir errores durante el cálculo de los modelos. En definitiva, se trata de un paso importante para el modelaje de los datos. La particularidad que tiene es que el escalado se produce teniendo en cuenta los valores mínimo y máximo de cada característica. En la Figura 43, se puede apreciar cómo primero se crea una copia del dataframe, posteriormente se seleccionan las columnas a las que se va a aplicar el escalado excluyendo el identificador de producto ya que no es una característica numérica y por tanto no se le puede aplicar el escalado. Se define el escalador, y se aplica la transformación de las columnas seleccionadas almacenándose el resultado en el dataframe creado previamente mediante una copia. El resultado es el que se puede apreciar en la Figura 43 dónde se puede apreciar cómo ha dado resultado el escalado.

```
# Creamos una copia del DataFrame original
df_normalized = df_modelings.copy()

# Seleccionamos las columnas a normalizar excluyendo 'ProductId'
columns_to_normalize = [col for col in df_modelings.columns if col != 'ProductId']

# Aplicamos MinMaxScaler
scaler = MinMaxScaler()
df_normalized[columns_to_normalize] = scaler.fit_transform(df_modelings[columns_to_normalize])

# Mostramos el DataFrame normalizado
df_normalized.head()
```

	Label_Encoded	ProductId	Component_1	Component_2	Component_3	Component_4	Component_5
0	0.0	B006N3IE6A	0.374863	0.574886	0.436204	0.334809	0.295144
1	0.0	B004FGWU9O	0.572844	0.728678	0.464442	0.563368	0.591214
2	0.0	B004FGWU9O	0.417712	0.469066	0.434388	0.391862	0.333373
3	0.0	B004FGWU9O	0.286153	0.526080	0.435745	0.477068	0.483409
4	0.0	B004FGWU9O	0.265682	0.457134	0.438089	0.405590	0.360529

Figura 43. Aplicación de MinMaxScaler

Llegados a este punto, ya se dispone del conjunto de datos limpio, analizado y procesado convenientemente para poder pasar a la siguiente fase del estudio, qué es el modelado de los datos.

Modelado de los datos

En este punto, se va a proceder al modelaje de los datos aplicando distintos modelos para la clasificación de textos y proceder así al análisis de sentimiento de los mismos. En una primera fase se realizará la división de entrenamiento y test de una manera particular a la que se suele realizar normalmente en la literatura de ciencia de datos, posteriormente se hablará de los distintos algoritmos con los que se pueden modelar los datos de los que se dispone, además de explicar las particularidades y limitaciones que se han observado durante los estudios, y

finalmente se discutirán los distintos rendimientos obtenidos de estos datos en función del modelo aplicado seleccionando aquel que mejor se ajuste a los datos de los que se dispone.

Se hará énfasis, en el rendimiento obtenido de estos datos en base al entorno con las limitaciones que se dispone, esto es importante ya que en un entorno empresarial siempre puede haber limitaciones de algún tipo, lo que hace que haya que aplicar estrategias para hacer eficiente los modelos y además utilizar los modelos que mejor se ajusten a los datos en base a distintas métricas como puede ser el tiempo de entrenamiento, los recursos disponibles…

En una primera instancia, se realizarán estudios sobre algoritmos ya conocidos cómo Random Forest, XGBoost, Naive Bayes… y posteriormente, se hará uso de HuggingFace, qué posee una plataforma de código abierto que permite a desarrolladores de todo el mundo, acceder y utilizar modelos de NLP pre-entrenados.

División de los conjuntos de entrenamiento y test

El primer paso antes de realizar el modelaje de los datos, es la separación de los conjuntos de entrenamiento y test. Como ya se ha ido deslizando anteriormente, la separación de los datos en este proyecto para la creación de estos dos conjuntos, Sara de una manera distinta al que se suele realizar en buena parte de la literatura de ciencia de datos. Normalmente la práctica que se suele utilizar es la separación aleatoria de muestras en los conjuntos de test y entrenamiento, de forma que en el entrenamiento no haya muestras del conjunto de test, y que durante la etapa de evaluación se pueda validar el entrenamiento que se ha realizado, con muestras que no haya visto el modelo durante el entrenamiento.

En este caso el enfoque de la separación va a ser el mismo, pero la forma de hacerlo va a ser ligeramente diferente. En este caso, en lugar de tomar todo el conjunto de datos y realizar una separación aleatoria, se realizará una separación de los datos en los distintos conjuntos teniendo en cuenta el identificador *ProductId*, es decir, se tomará un conjunto de productos para el entrenamiento, y otro conjunto distinto de productos para la validación. Al final la separación en los dos conjuntos atenderá a la filosofía de que se entrenará con una serie de productos el modelo, y después se validará el modelo con otro subconjunto totalmente distinto de productos. Por ejemplo, puede darse el caso que en el conjunto de entrenamiento se tenga un producto de café, mientras que en el conjunto de test se pueda encontrar un producto de patatas fritas.

Con este enfoque, se consigue que Amazon según vaya recopilando reseñas de nuevos productos, pueda validar dicho producto con las reseñas que ya dispone del resto de productos que tiene a la venta en su portal web. De esta forma se puede obtener una puntuación general del producto en base a otros, y no con reseñas aisladas de un producto en particular.

```
# Separamos características y etiquetas
X = df_modelings.drop(columns=['Label_Encoded', 'ProductId'])  # Características
y = df_modelings['Label_Encoded']  # Etiquetas
product_ids = df_modelings['ProductId']  # Para división

# Dividimos en conjuntos de entrenamiento y prueba según el ProductId
product_ids_train, product_ids_test = train_test_split(
    product_ids.unique(), test_size=0.2, random_state=42
)

# Filtramos las filas basandonos en ProductId
train_indices = df_modelings['ProductId'].isin(product_ids_train)
test_indices = df_modelings['ProductId'].isin(product_ids_test)

X_train, X_test = X[train_indices], X[test_indices]
y_train, y_test = y[train_indices], y[test_indices]

print("Conjunto de entrenamiento:", X_train.shape, "Conjunto de prueba:", X_test.shape)
```

```
Conjunto de entrenamiento: (70654, 100) Conjunto de prueba: (19346, 100)
```

Figura 44. Creación de los conjuntos de test y entrenamiento

En la Figura 44, se puede observar el proceso de división de los datos en los distintos conjuntos. En primer lugar, en las variables "x", se contendrán las características o variables independientes eliminando las columnas *Label_encoded* y *ProductId*. Esto es en primer lugar, porque las variables "y" contendrán las etiquetas que se corresponden con la variable *Label_encoded*, ya que será la variable objetivo. Finalmente, *products_ids* será la variable que extrae los identificadores únicos de los productos para realizar la división por identificadores.

A continuación, se seleccionan de manera aleatoria los identificadores de productos únicos para los dos conjuntos, 1 para entrenamiento (*product_ids_train*) y otro para test (*product_ids_test*). Al parámetro *test_size* se le asigna un valor de 0.2, de forma que el 20% de los productos Irán al conjunto de prueba, y el 80% de los productos restantes Irán al conjunto de entrenamiento. Posteriormente a esta división, se van identificando los índices de las filas correspondientes a los productos seleccionados para el entrenamiento y prueba a fin de obtener las filas completas. Una vez obtenidos dichos índices, simplemente se asignan a los conjuntos de entrenamiento y test, las filas correspondientes a dichos índices. Cómo se puede observar en el resultado final, se obtienen alrededor de 70000 filas para el conjunto de entrenamiento y cerca de 20000 para el conjunto de prueba.

Hay que destacar que el número de columnas se corresponde únicamente al de los textos ya procesados en valores numéricos con las transformaciones que se han realizado anteriormente, y en el caso de la variable objetivo se tendrá la variable *Label_Encoded.* Nótese que para el entrenamiento no se dispondrá del identificador de producto, ya que ha sido conservado hasta este punto para realizar la división de los datos única y exclusivamente de esta manera. Este es el fin principal con el que ha sido conservado el identificador de producto, ya que de no realizar la división de esta manera atendiendo a clases de productos, la división se podría haber realizado de una manera aleatoria siempre que el conjunto de datos estuviera balanceado en cuanto a reseñas, puntuaciones… Aunque también ha servido previamente para ver la relación que tiene respecto a las reseñas de los productos, y así se ha podido conseguir un análisis más completo.

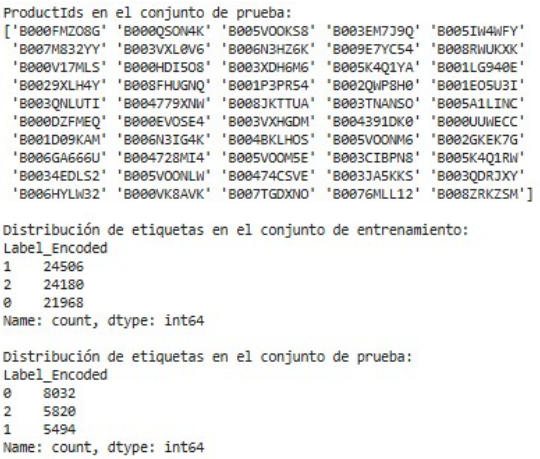


Figura 45. ProductId en el conjunto de test y distribución de etiquetas en los conjuntos

En la Figura 45, se puede observar los productos que van a conformar el conjunto de prueba, y también la distribución de etiquetas qué hay tanto en el conjunto de entrenamiento y en el conjunto de prueba. Se puede observar, que la distribución de etiquetas es bastante equitativa si bien es cierto que en la clase 0, correspondiente a las reseñas negativas, es ligeramente diferente que en el caso de las otras dos clases. Pero en general la división, es bastante equitativa para las 3 clases.

Una vez se tienen los conjuntos de entrenamiento y test se puede proceder a realizar los modelados correspondientes y comprobar cómo se ajustan los datos de los que se dispone a los distintos modelos.

Regresión Logística

En cualquier problema de clasificación, siempre se recomienda comenzar el modelaje de los datos mediante una Regresión Logística. Esto es, por qué se trata de un algoritmo muy simple, ya que en esencia se trata de un modelo lineal y esto facilita interpretar la relación entre las características y la posibilidad de pertenecer a una clase determinada. Además, se trata de un modelo rápido de configurar y entrenar lo que permite obtener un punto de referencia de partida. Además, computacionalmente consume menos recursos que otro tipo de algoritmos como pueden ser redes neuronales o modelos compuestos.

Para implementarla, se define una malla de búsqueda (grid_search), dónde se especifican distintos parámetros. En la Figura 46, se definen distintos hiperparámetros para intentar obtener la mejor combinación posible, que permita sacar el máximo rendimiento a los datos que se poseen. En primer lugar, se define el parámetro *class_weight*, como balanceado que, aunque se sabe que el conjunto de datos ha sido balanceado previamente puede ayudar a mejorar el rendimiento del algoritmo en caso de encontrar alguna desavenencia. Además, hay distintos tipos de penalización, aunque sea optado por L2 (Regularización de Ridge), para evitar el sobre ajuste a limitar la magnitud de los coeficientes del modelo. L2, es compatible con distintos *solvers* que se han probado para intentar obtener el mejor rendimiento. En una primera instancia, se probó con *elasticnet*, pero los resultados no fueron nada esperanzadores. En su caso se cambió por '*sag'* ya que funciona únicamente con la regularización de Ridge. Además, se configura un máximo de iteraciones elevado que permita hacer converger al modelo. Las primeras pruebas que se realizaron, fueron con un máximo de 5000 iteraciones, algo ínfimo que hacía que el modelo no convergiese.

```python
# Configuración de opciones de hiperparametros
param_grid = {
    'C': [0.0001, 0.001, 0.01, 0.1],  # Regularización inversa
    'penalty': ['l2'],  # Penalización
    'solver': ['lbfgs', 'saga', 'sag'],  # Solver eficiente
    'max_iter': [20000],  # Máximo de iteraciones
    'class_weight': ['balanced']
}

log_reg_model = LogisticRegression(random_state=42, multi_class='ovr', n_jobs=-1)

# StratifiedKFold para la validación cruzada
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Búsqueda de hiperparámetros
grid_search = GridSearchCV(
    estimator=log_reg_model,
    param_grid=param_grid,
    cv=stratified_kfold,  # Utilizar StratifiedKFold
    scoring='f1_weighted',
    n_jobs=-1
)


grid_search.fit(X_train, y_train)
```

Figura 46. Configuración de Regresión Logística.

Por tanto, se fue incrementando el número de iteraciones para intentar que convenciese el modelo. El parámetro se llegó a incrementar hasta las 100000 iteraciones, algo que con este volumen de datos y los recursos disponibles hizo que se reiniciase la sesión por desbordamiento de la RAM. Aun así, los resultados tampoco fueron nada buenos cómo se verá a continuación.

Además, se define validación cruzada estratificada, qué es una variación de la validación cruzada que ayuda que la distribución de las clases en cada subconjunto o folds, sea parecido a la distribución de clases que hay en el conjunto completo. De nuevo esto es útil sobre todo cuando se tiene un problema de clasificación con clases desbalanceadas, pero como se ha analizado anteriormente, los conjuntos de datos de cada clase positiva, negativa o neutra estaban muy balanceados. Sin embargo, se optó por ir un poco más allá además de utilizar la validación cruzada, hacerlo con esta variante para intentar mejorar los resultados. En la Figura 46, se puede apreciar la mejor combinación posible de hiper parámetros que se ha conseguido en el entorno con las limitaciones ya comentadas.

Con una regularización tan baja, puede decir que el modelo tiene mucha penalización para evitar sobreajustes, y puede penalizar el ajuste en datos complejos. Se aprecia también cómo el solver de la mejor opción es “sag”, que se trata de un solver bueno en grandes conjuntos de datos.

Si se atiende a las métricas, se puede apreciar que la matriz de confusión muestra distintos errores, ya que la clase 0 es confundida en su mayoría por la clase 1 con 4085 instancias, la clase 1 tiene confusiones importantes con las otras dos clases, con la clase 0 con 1905 instancias, y con la clase dos con 1358 instancias. Asimismo, la clase dos se confunde principalmente con la clase 1 aunque también, pero en menor proporción con la clase 0. Lo que viene a decir esto es que el modelo tiene grandes dificultades para distinguir entre las distintas clases por lo que indica que las características pueden no distinguir lo suficientemente bien.

```
Mejores hiperparámetros: {'C': 0.0001, 'class_weight': 'balanced', 'max_iter': 20000, 'penalty': 'l2', 'solver': 'sag'}

Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.26      0.17      0.20      8032
           1       0.26      0.38      0.30      5494
           2       0.35      0.37      0.36      5820

    accuracy                           0.29     19346
   macro avg       0.29      0.30      0.29     19346
weighted avg       0.29      0.29      0.28     19346

Matriz de Confusión:
[[1347 4085 2600]
 [2074 2062 1358]
 [1758 1905 2157]]

Curva AUC-ROC:
AUC-ROC Score: 0.49
```

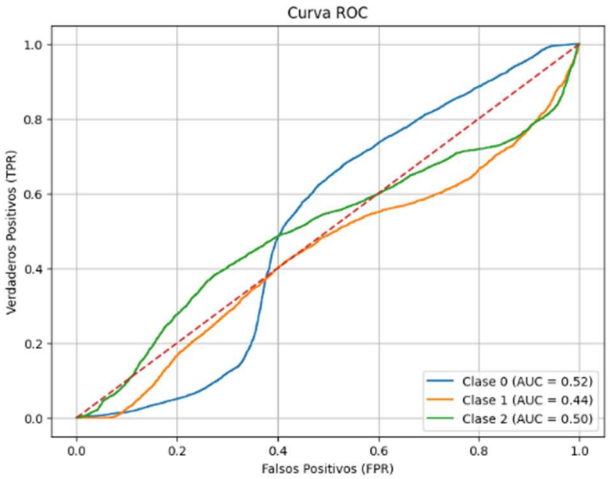


Figura 47. Resultados de la mejor opción para la validación cruzada

En cuanto a la curva AUC-ROC la puntuación obtenida es de 0.49, lo que es un valor muy bajo y muy malo ya que el modelo no es mejor que un clasificador aleatorio. Esto indica que el modelo no está captando los patrones que hay en los datos.

Intentar mejorar esto es difícil, ya que por un lado se intentó incrementar la regularización para que el modelo se ajuste mejor a los datos, pero fue insuficiente. También se probó con validación cruzada sin estratificar, pero los resultados fueron aún más negativos. Esto hace pensar que este modelo no es idóneo para los datos que se poseen para realizar este estudio, por lo que es interesante probar otro tipo de modelos.

Naive Bayes

Este algoritmo de clasificación basado en el Teorema de Bayes, es eficiente y simple de implementar, por lo que puede ser interesante probarlo para ver cómo se comporta. El modelo escogido es GaussianNB como se puede apreciar en la Figura 48, que asume que los datos siguen una distribución normal. Se hace uso de validación cruzada con 3 folds, para ser más eficientes en el entrenamiento, y la regularización de la varianza, es el parámetro que se varía.

```python
# Ajuste de la regularización de la varianza
param_grid = {'var_smoothing': [1e-30,1e-25,1e-20,1e-15]}

# Naive Bayes con GridSearchCV
naive_Bayes_Model = GaussianNB()
grid_search = GridSearchCV(estimator=naive_Bayes_Model,
                           param_grid=param_grid,
                           cv=3,  # Número de folds en validación cruzada
                           scoring='f1_weighted',
                           n_jobs=-1)

grid_search.fit(X_train, y_train)
```

Figura 48. Definición del modelo Naive Bayes con sus parámetros correspondientes

En cuanto a los resultados, al fijarse en la Figura 49, en primer lugar, se aprecia una exactitud del 37% lo que quiere decir que el modelo acierta aproximadamente 1/3 de las predicciones, lo que hace un modelo bastante pobre. En promedio la Precisión es de 0.35, con un Recall de 0.41 y un F1-Score de 0.36 lo que implica un rendimiento bajo y también desigual entre clases ya que yendo clase por clase hay algunas en las que el desempeño es ligeramente superior a otras, pero en general sigue siendo muy pobre. Para la clase cero, yendo valor por valor el modelo tiene problemas serios para identificar correctamente esta clase, por ejemplo, el Recall que es del 11%, implica que solo se predice correctamente 1 de cada 10 muestras de cada clase lo que es ínfimo. La clase 1 es la que mejor reconoce el modelo dentro de lo malo qué es el modelo para ajustarse a estos datos, y la clase dos tiene un rendimiento ligeramente mejor que la clase cero pero que tenga un Recall del 60% indica que este modelo es más propenso a identificar muestras en esta categoría.

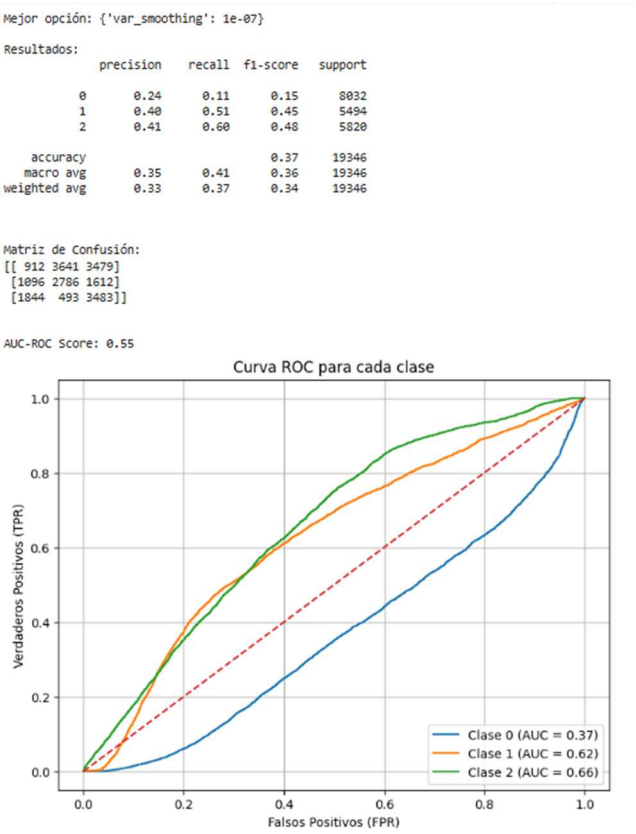


Figura 49. Mejor rendimiento obtenido con Naive Bayes

En cuanto a la matriz de confusión en la clase cero muchos ejemplos se confunden con las clases 1 y dos lo que dice que el modelo tiene altas dificultades para distinguir esta clase. Hay que recordar que la clase cero son las reseñas negativas. En cuanto a la clase 1 se reconoce mejor que la clase cero, pero hay muchas muestras mal identificadas. Este comportamiento se aprecia también en la clase dos. En cuanto a la Curva ROC-AUC, la puntuación es de 0.55 y esto dice que el modelo tiene poca capacidad para distinguir entre clases.

En cuanto a la curva ROC, se aprecia cómo el rendimiento del modelo varía significativamente entre las distintas clases ya que para la clase 0, AUC es 0.37, y para las clases 1 y 2 están por encima de 0.5, concretamente por encima de 0.6, lo que para estas últimas dos clases el rendimiento es moderado, pero está muy lejos de ser óptimo. Destacar en este punto que también se probaron otras variantes como BernouilliNB o MultinomialNB, con su respectiva

variación de parámetros, pero los resultados fueron aún peores, por lo que se desestimó por no lograr ajustarse a los datos convenientemente.

Con esto, se aprecia que este modelo tampoco es idóneo para los datos de los que se disponen, lo que hace pensar que quizás modelos tan simples no sean del todo idóneos para poder ajustarse a la complejidad y características de los datos de este estudio, por lo que sería interesante probar con modelos compuestos.

Random Forest

Cómo se sabe esta técnica es una combinación de distintos árboles de decisión, de forma que es un modelo compuesto que permite aumentar la precisión y a su vez reduce el sobreajuste ya que se combinan los resultados de muchos árboles para mejorar la capacidad de generalización. Además, es bueno frente al ruido lo que implica que, si existen outliers, será un modelo robusto y al combinar también múltiples árboles permite que sea más preciso.

```
# Configuración de valores para buscar el optimo
param_grid = {
    'n_estimators': [10, 20, 50],          # Número de árboles
    'max_depth': [10, 20, 50],           # Profundidad máxima
    'min_samples_split': [1, 5],
    'min_samples_leaf': [1, 2, 5],
    'class_weight': ['balanced']     # Balancear clases
}

random_forest_model = RandomForestClassifier(random_state=42, n_jobs=-1)

# Búsqueda de hiperparámetros
grid_search = GridSearchCV(
    estimator=random_forest_model,
    param_grid=param_grid,
    cv=3,
    scoring='f1_weighted',
    n_jobs=-1
)

grid_search.fit(X_train, y_train)
```

Figura 50. Definición de parámetros de RandomForest

En cuanto a la definición de parámetro definición de parámetros en random Forest, se aprecia en la Figura 50 que, en primer lugar, se coloca distintas opciones de número de árboles, con distintas opciones de máxima profundidad de los árboles, se especifica un mínimo demuestras por cada hoja, así como el número mínimo de muestras para dividir un nodo.

Teóricamente un mayor número de árboles puede aportar mayor precisión, pero puede ser más lento a la hora de entrenar, árboles más profundos permiten captar patrones más complejos, pero se podría correr el riesgo de sobreajuste, el número de muestras para dividir un nodo cuanto mayor alto es los nodos solo se dividen si hay suficientes datos por lo que si no hay suficientes datos los árboles son más simples. Finalmente, el número mínimo de muestras por hoja cuanto mayor sea asegura que cada hoja tiene más información lo que hace árboles más generalizados. Sin embargo, una cosa es la teoría y otra es la práctica por lo que hay que ver el comportamiento de estos datos sobre el modelo.

```
Mejor opción: {'class_weight': 'balanced', 'max_depth': 50, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 50}

Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.96      0.65      0.78      8032
           1       0.61      0.98      0.75      5494
           2       1.00      0.85      0.92      5820

    accuracy                           0.81     19346
   macro avg       0.85      0.83      0.81     19346
weighted avg       0.87      0.81      0.81     19346


Matriz de Confusión:
[[5251 2777    4]
 [  79 5408    7]
 [ 149  749 4922]]

Curva AUC-ROC:
AUC-ROC Score: 0.98
```

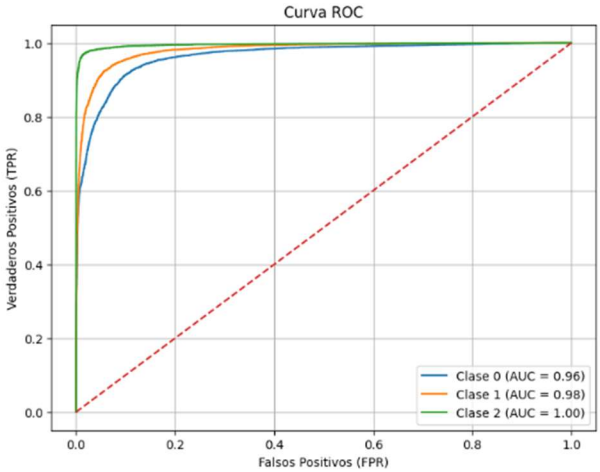


Figura 51. Resultados del modelo Random Forest con la mejor combinación de parámetros posible

En la Figura 51, a simple vista se puede observar cómo los resultados son excelentes. En primer lugar, estos resultados se han obtenido con un número de árboles igual a 50 que, aunque es un número no muy elevado equilibra la rapidez del entrenamiento y a la vez mejorando la precisión, y como se aprecia parece suficientemente bueno este valor. El número de muestras por hoja es al menos 5 y cada nodo debe de tener al menos 5 muestras para dividirse, esto indica que los árboles no son muy complejos y por tanto evita el sobreajuste, lo que es una gran noticia. Los árboles como máximo tiene una profundidad de hasta 50 niveles lo que permite obtener o capturar relaciones complejas en los datos.

Si se realiza un análisis por clase, la clase 0 (*negative*) tiene una precisión del 96% lo que identifica prácticamente en su totalidad las muestras de esta clase. El Recall es del 65% por lo que solo el 65% de las muestras de la clase cero son detectadas correctamente, y el F1- Score es del 78% lo que muestra un rendimiento razonablemente bueno. En la clase 1 la precisión es del 61% qué es baja por lo que indica que varias predicciones de esta clase son incorrectas, el Recall es del 98% lo que hace que se capture perfectamente las instancias de esta clase, y F1-Score es también bastante bueno. Para la clase 2 la precisión es del 100%, Recall del 85% y F1-Score del 92%, lo que dice que el rendimiento para esta clase es excelente.

En cuanto a la matriz de confusión, en la clase 0 muchas instancias se clasifican como clase 1, por lo que puede decir que se parece a la clase 1. Mientras en las otras dos clases hay pocas confusiones, lo que dice que este modelo funciona muy bien para estas dos clases. En la curva, ROC-AUC Score, el resultado es de 0.98 que es excelente, lo que dice que este modelo distingue bien entre clases. Por cada clase, no hay ninguna puntuación que baje del 0.96, siendo en este caso la clase 0 la peor.

Este podría ser uno de los modelos a seleccionar como idóneo para este estudio, ya que el rendimiento es sobresaliente, pero se debe de probar más opciones para seleccionar la mejor.

Gradient Boosting

Se trata de una técnica de aprendizaje supervisado que se caracteriza por combinar múltiples arboles de decisión secuencialmente. Esto hace que cada intenta corregir los errores cometidos por el anterior, mediante la minimización de la función de perdida log-loss.

```
fixed_params = {
    'learning_rate': 0.01,
    'max_depth': 10,
    'n_estimators': 700,
    'subsample': 0.8,
    'colsample_bytree': 0.8
}
```

Figura 52. Parámetros para XGBoost

Hay que destacar la alta velocidad de entrenamiento que tiene este modelo, lo que lo hace eficiente dada la falta de recursos que se tiene en este proyecto con la cuenta gratuita de Colab. Se toma como parámetros más destacados, los indicados en la Figura 52, con una tasa de aprendizaje (*learning_rate*), que es lo que aporta cada árbol al modelo final, de 0.01. Cuanto menor sea, aprende más lento y el entrenamiento por ende tarda más, pero mejora la generalización. La profundidad máxima de los árboles indica lo máximo de complejos que pueden ser, y en este caso es 10. El número de estimadores indica el número de árboles a construir en el modelo, y cuanto mayor es este valor, mejor es para detectar valores complejos. En este caso es de 700. Finalmente se tiene la proporción de muestras para construir cada árbol, lo que introduce aleatoriedad en el entrenamiento evitando el sobreajuste, dado por 0.8.

En la figura 53, se observa como en las métricas generales, se tiene un Accuracy de 0.83 lo que indica qué clasifica correctamente el 83% de las muestras lo cual es un clasificador multiclase bastante sólido. La curva ROC tiene un AUC de 0.98, lo que indica una gran capacidad de este modelo para distinguir entre las distintas clases. Además, con un 0.85 de Recall, y 0.84 de F1-Score, se tiene un rendimiento entre clases bastante equilibrado.

Si se hace un análisis por clases, destaca en el plano negativo en la clase cero un Recall de 0.67, lo que indica que solo identifica correctamente el 67% de las muestras de la clase cero. En el resto de campos destaca notablemente en todos los aspectos. En la matriz de confusión, se puede apreciar cómo hay confusiones entre la clase 0 y la clase 1 ya que la clase cero se confunde frecuentemente con la 1, y la clase 2 le pasa el mismo problema, aunque en menor medida.

```
Resultados:
              precision    recall  f1-score   support

           0       0.97      0.67      0.79      8032
           1       0.63      0.98      0.77      5494
           2       1.00      0.91      0.95      5820

    accuracy                           0.83     19346
   macro avg       0.87      0.85      0.84     19346
weighted avg       0.88      0.83      0.83     19346


Matriz de Confusión:
[[5347 2683    2]
 [  82 5404    8]
 [  98  443 5279]]

AUC-ROC Score: 0.98
```

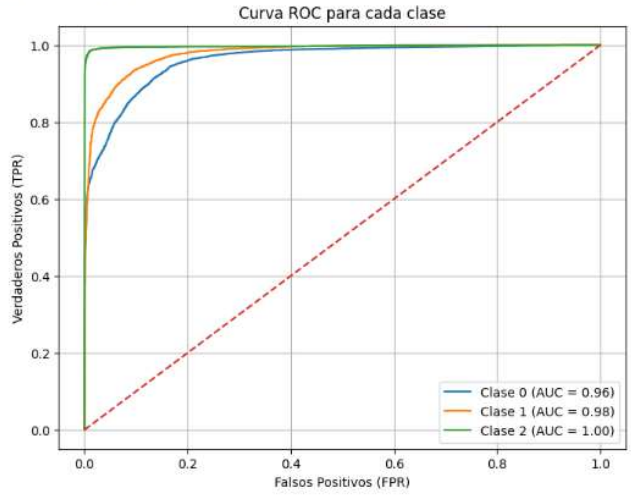


Figura 53. Resultados de XGBoost obtenidos

Este modelo también tiene un gran rendimiento frente a los datos, por lo que en la discusión final es otro de los candidatos a elegir.

Redes Neuronales

A continuación, se estudiará el rendimiento de redes neuronales para clasificación, por lo que se hará uso del framework Tensorflow muy útil para la construcción de redes neuronales de una manera simple, en combinación con Keras que facilita la construcción y el entrenamiento de modelos en combinación con tensorflow.

Para la definición de la red neuronal, se debe de importar *Sequential* y con él se puede definir el modelo como se muestra en la Figura 54. Con *Sequential*, las capas se van construyendo una detrás de otra. Se definen 8 capas, en la que la capa de entrada tiene 512 neuronas y le sigue otra capa con una tasa de descarte (Dropout) de 0.5. Esta característica apaga el 50% de las neuronas para evitar sobreajustes del modelo. Posteriormente hay una tercera capa en la que se reduce a la mitad del número de neuronas a 256, y dado que hay menos neuronas en esta capa, en la cuarta capa, la tasa de descarte se puede bajar un poco, concretamente a 0.4. Se va realizando así sucesivamente hasta una cuarta capa de 64 neuronas. Es importante destacar que en las funciones de activación se emplea *relu*, salvo la última capa, de salida, que se emplea *softmax* ya que es la función de activación más idónea para clasificación. La función de activación *relu* hace que aprendas relaciones complejas, mientras que *softmax* ayuda a determinar la clase, asignando una probabilidad a cada clase.

Posteriormente para compilar el modelo, se define un optimizador Adam qué es muy usado en este tipo de redes ya que es eficiente y tiene una buena capacidad para ajustarse a distintos problemas. Se define una tasa de aprendizaje muy pequeña para que vaya aprendiendo en pasos muy pequeños y encontrar el valor óptimo de la manera más precisa. La función de pérdidas, se

toma como *sparse_categorical_crossentropy* ya que es óptima para los problemas de clasificación de múltiples clases. Después, se define el parámetro *early_stopping* que permite detener el entrenamiento cuándo las pérdidas de validación no mejoran en al menos 12 épocas. Finalmente se define una reducción adaptativa de la tasa de aprendizaje, qué es una técnica usada para reducir la tasa de aprendizaje cuando el rendimiento del modelo deja de mejorar durante el entrenamiento. Por tanto, cuando se detecta que el modelo deja de mejorar se reduce automáticamente la tasa de aprendizaje para intentar afinar y ajustar el modelo con mayor precisión.

```
# Definición del modelo
model = Sequential([
    Dense(512, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.4),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(len(np.unique(y_train)), activation='softmax')
])

# Compilación del modelo
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',  # Para etiquetas enteras en problemas multiclase
              metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss',  # Supervisar la pérdida de validación
                               patience=12,         # Esperar 12 épocas sin mejora
                               restore_best_weights=True)  # Restaurar los mejores pesos

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)
```

Figura 54. Definición de la red neuronal usada para obtener los mejores resultados posibles

Cabe destacar que se probaron múltiples combinaciones para intentar mejorar el rendimiento, aunque la que se puede ver en la Figura 54 es la que finalmente se ha seleccionado gracias a un compromiso entre gran rendimiento y eficiencia frente a las limitaciones de memoria RAM. Se probó con más capas, llegando hasta una configuración de 10 capas donde la de entrada tenía 1024 neuronas, pero que sin embargo eso no mejoró el rendimiento de la arquitectura que se acabó seleccionando. También se probó con más capas cercanas a la salida, pero tampoco dio el suficiente resultado. Otra de las variaciones fue cambiar el número de neuronas por capa, es decir, se realizaron distintas permutaciones de valores para estudiar el comportamiento de la red en base a estos datos.

En el entrenamiento, se comenzó con un número de épocas de 30, lo que era extremadamente bajo por lo que se fue incrementando paulatinamente hasta llegar a las 150 pero que tampoco se llegó a consumir estas 150 debido a que antes entró en juego el parámetro *early_stopping* ajustando el modelo mucho antes. Cabe recordar, que las épocas es el número de veces que el modelo consume los datos de manera completa. También es importante destacar, que el *batch_size*, qué es el parámetro que indica el número de muestras que el modelo procesa antes de actualizar los pesos durante el entrenamiento, se fue reduciendo hasta tener un tamaño de 64, ya que con 128 era demasiado tamaño para la cantidad de memoria RAM que tiene Colab en su versión gratuita. Esto influye en que el entrenamiento tardará un poco más en realizarse, pero puede manejar los datos sin ningún problema.

```
Resultados:
              precision    recall  f1-score   support

           0       0.94      0.78      0.85      8032
           1       0.73      0.96      0.83      5494
           2       0.80      0.76      0.78      5820

    accuracy                           0.82     19346
   macro avg       0.82      0.83      0.82     19346
weighted avg       0.84      0.82      0.82     19346

Matriz de Confusión:
[[6235  812  985]
 [ 150 5253   91]
 [ 239 1168 4413]]
AUC-ROC Score: 0.94
```

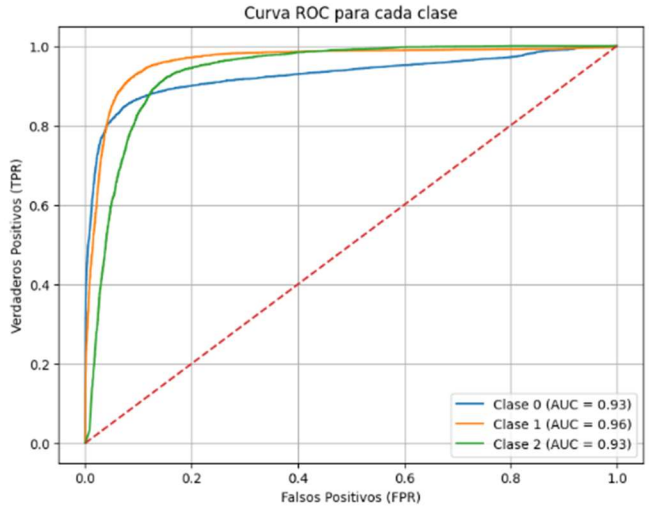


Figura 55. Resultados de la red neuronal de clasificación

En la Figura 55, se pueden apreciar los resultados obtenidos a través de esta arquitectura de la red neuronal. En cuanto a las métricas generales, se obtiene que el modelo clasifica correctamente el 82% de las muestras totales, lo que indica un buen resultado, y el Recall y F1-Score obtienen respectivamente 0.84 y 0.82. un buen balanceo entre clase, lo que denota un buen balanceo entre clases. El área bajo la curva (AUC) es de 0.94, lo que hace a este modelo excelente para distinguir entre las clases.

Si se analiza en cada clase, en la clase 0 tiene una precisión del 94%, también se indica un buen balance entre precisión y Recall, además de tener un Recall de 0.78 lo que quiere decir que 4 de cada 5 muestras de esta clase fueron identificadas correctamente. En la clase 1 la precisión es más baja ya que es del 73%, lo que puede indicar confusiones a la hora de determinar si la muestra es de clase 1 o de otra clase, pero se obtiene un Recall de cero 96 qué es excelente, mostrando una capacidad alta para identificar las verdaderas muestras de la clase 1. F1-Score es de 0.83 que sigue teniendo un buen balanceo entre precisión y Recall. En la clase dos la precisión baja el 80% con lo que está entre las dos clases en esta característica, el Recall indica que este modelo identifica el 76% de las muestras de la clase de 2 correctamente, pero baja un poco también en F1-Score lo que hace que el rendimiento en esta clase sea un poco peor que en las otras dos clases.

En la matriz de confusión, se puede observar cómo hay confusiones importantes entre la clase cero y la clase dos lo que indican que podría tener características similares mientras que la clase 1 tiene un buen rendimiento en comparación con las otras dos clases. Esto también se puede observar gráficamente en la Figura 55 y en las curvas de cada una de las clases. Indicar también que el área bajo la curva de cada una de las clases es excelente, mostrando el mejor caso para la clase 1, pero muy bueno también para las otras dos clases.

Como conclusión general hasta este punto, se observa que los modelos simples no han sido capaces de captar las complejidades que poseen estos datos, y es ahí donde modelos compuestos o redes neuronales han destacado notablemente mostrando un rendimiento muy superior con una capacidad de detección de cada clase muy notable.

A continuación, se va a explorar una última posibilidad dentro del aprendizaje supervisado, qué es el problema que se está tratando en este proyecto, y es la utilización de Transfer Learning, qué es utilizar redes neuronales ya entrenadas, pero cuyos pesos ya ajustados en las capas ocultas (las capas intermedias de una red neuronal) se congelan de tal manera que no se permite ajustarse durante el entrenamiento, ya que en la fase de entrenamiento se ajustan los pesos correspondientes a las capas superficiales de salida, para que la red neuronal tarde mucho menos en entrenar para los datos que se deseen. En este caso, se promete aprovechar la red neuronal preentrenada para hacer un ajuste fino a los datos de los que se disponen en este proyecto. Esto se realizará mediante HuggingFace.

HuggingFace

Se trata de una empresa y plataforma que se ha convertido en un estándar de facto en los campos de inteligencia artificial y concretamente en procesamiento de lenguaje natural o NLP. Ofrece muchas herramientas y bibliotecas diseñadas para cualquier investigación, empresa… Tal y como afirma Navarro S. (2024).

Cómo característica más importante, ofrece la librería *Transformers,* que se trata de una biblioteca de código abierto que permite utilizar modelos pre entrenados basados en arquitecturas de última generación como pueden ser BERT, GPT-2… esto permite realizar multitud de tareas, ya que son modelos específicamente diseñados para trabajar con textos, y entre las tareas que permite realizar se pueden destacar la clasificación de texto, traducción automática, respuesta a preguntas, clasificación de imágenes, procesamiento de audio… lo que la hace una biblioteca muy potente para diversas aplicaciones.

Además, también dispone de Hugging Face Hub, qué es un repositorio en la nube que permite a los usuarios subir, compartir, descargar modelos… al estilo de GitHub. Dispone también de librerías para facilitar la carga y procesamiento de conjuntos de datos comunes para Machine Learning y NLP llamada *datasets*.

Por tanto, será de gran utilidad para comprobar el ajuste de los datos que se disponen en este proyecto a distintos modelos aprovechando la transferencia de aprendizaje en las redes neuronales. Es importante destacar también, que con HuggingFace se puede utilizar aprendizaje supervisado que es el caso que ocupa este proyecto, ya que, hay otras librerías como word2vec o GloVe, que se utilizan en el aprendizaje no supervisado.

Para la utilización de esta librería, se puede utilizar tanto Tensorflow como Pytorch (Librería de IA de Meta). En este caso, como ya se ha utilizado anteriormente Tensorflow, se hará uso del mismo framework para la construcción del flujo de datos necesario para poder entrenar con esta librería. De nuevo la mayor dificultad a la hora de realizar la construcción, ha sido la escasez de memoria RAM, que como se explicará en la construcción del código, ha hecho que se tenga que realizar diversas operaciones para que el entorno no consumiese todos los recursos y reiniciarse la sesión por desbordamiento de la memoria RAM.

Procesamiento previo

Esta biblioteca trabaja directamente con los textos sin tokenizar, preprocesar… ya que la propia librería realiza un procesamiento específico de los textos dependiendo del modelo que se vaya a utilizar para poderlo ajustar de la mejor manera al modelo y sacar el mayor rendimiento de este. Por tanto, para cada modelo que se vaya a utilizar, se debe de procesar el texto de una manera determinada, puesto que cada modelo tiene su tokenizador determinado, por ejemplo.

Es por ello, que se deben de tomar los datos del conjunto original, ya que los textos los tratará de manera específica el modelo que se use para realizar el entrenamiento, tokenizando y procesando los datos de una manera determinada. En casos anteriores, por ejemplo, se ha visto que para poder realizar un entrenamiento de los datos hay que vector izar el texto ya que hay que transformar el texto a variables numéricas que los modelos puedan entender. En este caso no hace falta realizar ese procesamiento ya que se va a encargar el modelo en cuestión de realizarlo antes de realizar el entrenamiento pertinente.

El primer paso será tomar el subconjunto de datos que se obtuvo del conjunto de datos original de las reseñas de Amazon de 568000 filas por 10 columnas, dando como resultado a la muestra del conjunto de datos con el que se ha trabajado a lo largo del proyecto de 86500 filas y 10 columnas. Posteriormente, hay que realizar una serie de transformaciones que han sido descritas en apartados anteriores y que aquí se van a comentar, pero no con el mismo nivel de detalle con el que se ha descrito anteriormente, ya que a efectos prácticos se tratan de operaciones muy similares.

```
[ ] # Cargamos el csv_modelado
    df_loaded = pd.read_csv('/content/sample_reviews.csv')
    df_loaded.head(10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1603	B001RVFDOO	A1UV8UCBOYABKQ	deek	0	0	5	1329091200	yummy	These chips are wonderful. My kids and I all ...
1	1604	B001RVFDOO	A1MFIN31SXO81A	Foof	0	0	5	1329091200	Great Item	Great packaging, and item is just as described...
2	1605	B001RVFDOO	A1EMVE0E166IKY	G. Piller	0	0	5	1329004800	Excellent chips	Just to give a little background - I ordered t...
3	1606	B001RVFDOO	A2ZNXVTFNE6MQO	Lori "Lori"	0	0	5	1328918400	Perfect for WW followers	Got this item on a one-day-only sale on Amazon...
4	1607	B001RVFDOO	A30QMNHLS0YTKI	L. Ferguson "WineChick"	0	0	5	1328572800	Variety pack, packed lunches	I discovered Pop Chips awhile back and fell in...
5	1608	B001RVFDOO	AU0PC4RM01IOD	C. CARPENTER "part-time reader"	0	0	4	1328572800	Loved all the flavor except salt and pepper!	I really liked these chips as did my entire fa...
6	1609	B001RVFDOO	A2JKRWF7RCCI1	Tony's diamond	0	0	3	1328572800	chips	These popchips were something I wanted to try ...
7	1610	B001RVFDOO	A3PD8JD9L4WEII	Bron "Bron"	0	0	4	1328313600	Good as a sampler pack of the various flavors	I like these Pop Chips a lot, but not all the ...
8	1611	B001RVFDOO	A2HKGQ3OBD6Q23	D. Yang	0	0	5	1328054400	These tastes pretty good	These tastes good. If it's not fried, not bad ...
9	1612	B001RVFDOO	AX8KKN5A1PG9E	CintiBonnie "bonnie8"	0	0	2	1327622400	Didn't care for the variety selections	I bought two bags of PopChips at Whole Foods (...

Figura 56. Conjunto de datos de partida para HuggingFace

En la Figura 56, se puede observar el conjunto de datos de partida de 86500 filas y 10 columnas, sobre el que se van a realizar distintas transformaciones. En primer lugar, se eliminarán todas aquellas columnas que no sean necesarias para el estudio, conservando únicamente el identificador de producto, el resumen, el texto completo de la reseña, y la variable de puntuación de la reseña (*Score*).

A continuación se concatenan las columnas de resumen y texto de la misma forma que se hizo en secciones atrás, creando la columna *Review* y eliminando las columnas de texto y resumen, ya que una vez concatenadas no son de utilidad. Posteriormente se crea la columna *Label* en el dataframe que será la *feature* que indicará si el texto tiene un sentimiento positivo, negativo o neutro como ya se explicó también en secciones anteriores. De nuevo se elimina la columna *Score* ya que una vez creada la *feature*, no tiene utilidad, y se codifica la columna *Label* creando la columna *Label_Encoded*, para qué esos datos puedan ser manejados posteriormente.

Finalmente como se explicó en secciones anteriores se realiza un submuestreo de la clase mayoritaria que és la positiva, a la vez que se realiza un sobremuestreo de las clases minoritarias. En este punto sí que existe una diferencia con respecto a los procesos realizados con

anterioridad, y es que no se aplica la técnica SMOTE ya que en ese caso se requería la transformación del texto a características vectoriales y es algo que perjudica de cara a utilizar HuggingFace.

```
# Dividimos los datos en 3 clases, las mismas que en Label_Encoded
zero = df_hugging_under[df_hugging_under.Label_Encoded == 0]
one = df_hugging_under[df_hugging_under.Label_Encoded == 1]
two = df_hugging_under[df_hugging_under.Label_Encoded == 2]  # Clase mayoritaria

# Sobremuestreo de las clases minoritarias igualando al numero de muestras de la mayoritaria
zero_upsampled = resample(
    zero,
    replace=True,
    n_samples=len(two),
    random_state=42
)

one_upsampled = resample(
    one,
    replace=True,
    n_samples=len(two),
    random_state=42
)

# Combinamos todas las clases
df_hugging_balanced = pd.concat([zero_upsampled, one_upsampled, two]).sample(frac=1).reset_index(drop=True)
df_hugging_balanced.head()
```

Figura 57. Utilización de Resample para aumentar las muestras de las clases minoritarias

En la Figura 57, se muestra la forma de proceder en este caso, qué es con la función resample. Esta función simplemente lo que hace es multiplicar las muestras de las clases minoritarias hasta alcanzar el número de muestras de la clase mayoritaria, por tanto, no son creadas de manera sintética, sino que simplemente son duplicadas. Con esto, se llega a las mismas dimensiones de conjunto de datos que se tenía anteriormente, es decir 90000 filas por 3 columnas, correspondiente a las variables del texto ya concatenadas, el identificador de producto y finalmente *Label_Encoded*.

A continuación, se va a detallar la construcción, del flujo de datos necesario para poder realizar el entrenamiento de la manera más eficiente posible. Se ha tenido que realizar gran cantidad de operaciones para garantizar la eficiencia y evitar el desbordamiento de la RAM qué es un punto crítico en este entorno gratuito. Además, también se garantiza la división en entrenamiento y test por productos y no por reseñas, con lo que ese punto es exactamente igual que en casos anteriores.

Explicación del flujo de trabajo

El primer paso, cómo se aprecia en la Figura 58, se divide el conjunto de datos en dos subconjuntos de entrenamiento y test, teniendo en cuenta que la división se realiza teniendo en cuenta el identificador de producto, asegurando que en el conjunto de prueba no haya productos del conjunto de entrenamiento y viceversa.

```
# Se dividen los datos en función de ProductId para entrenamiento y test
product_ids_train, product_ids_test = train_test_split(
    df_hugging_balanced['ProductId'].unique(), test_size=0.2, random_state=42
)

# Filtramos las lineas en función de ProductId
train_indices = df_hugging_balanced['ProductId'].isin(product_ids_train)
test_indices = df_hugging_balanced['ProductId'].isin(product_ids_test)
df_train = df_hugging_balanced[train_indices]
df_test = df_hugging_balanced[test_indices]
```

Figura 58. Separación de conjuntos de entrenamiento y test para HuggingFace

A continuación, se carga el toque iniciador con el modelo deseado almacenado en una variable para poder adaptar el código de una manera más fácil entre distintos modelos como se aprecia en la Figura 59. Además, se define un máximo de tokens a 32 unidades mínimas de texto, esto

es importante ya que para manejar recursos limitados de memoria RAM se ha tenido que ir bajando este tamaño hasta uno que no desbordase la memoria. Por tanto, inicialmente se probó con 128, cómo desbordaba la memoria se bajó a la mitad, y de ahí al valor qué hace que no desborde la memoria que es 32.

```
model_token_name = "microsoft/MiniLM-L12-H384-uncased"

# Se carga el tokenizador con el modelo deseado, y una longitud maxima de
tokenizer = AutoTokenizer.from_pretrained(model_token_name)
tokenizer.model_max_length = 32
```

Figura 59. Definición del modelo y tokenizador

El siguiente paso es utilizar TFRecords. Como muestran López F. (2019) y Dimitre Oliveira (2023), la utilización de este formato, hace que sea imprescindible cuando se manejan volúmenes de datos muy elevados. Por tanto, para manejar esta cantidad de datos de manera eficiente se utiliza este tipo de ficheros. Esto permitirá ahorrar suficiente memoria para que no desborde la RAM y evitar que la sesión sea cerrada de manera automática.

Esto se aprecia en la Figura 60, que muestra una función para escribir los datos en este formato, de forma que convierte los textos y las etiquetas llamando a la función encode que se puede apreciar en la Figura 61, y posteriormente a la transformación, los escribe en un formato binario TFRecord. Para ello se incluye también, puntos importantes que realiza la biblioteca de HuggingFace y que salen de la operación de tokenización. Una es la representación tokenizada del texto, representado por la variable *input_ids*, la máscara que indica las posiciones válidas en los tokens dada por la variable *attention_mask*. Además, se incluye la etiqueta de la clase correspondiente, *label*.

```
# Función para escribir los datos en TFRecords y ahorrar RAM
def write_tfrecord(data, filename):
    with tf.io.TFRecordWriter(filename) as writer:
        for review, label in zip(data["Review"], data["Label_Encoded"]):
            input_ids, attention_mask, label = encode(review, label)
            features = {
                "input_ids": tf.train.Feature(int64_list=tf.train.Int64List(value=input_ids)),
                "attention_mask": tf.train.Feature(int64_list=tf.train.Int64List(value=attention_mask)),
                "label": tf.train.Feature(int64_list=tf.train.Int64List(value=[label])),
            }
            dataRecord = tf.train.Example(features=tf.train.Features(feature=features))
            writer.write(dataRecord.SerializeToString())

write_tfrecord(df_train, "train.tfrecord")
write_tfrecord(df_test, "test.tfrecord")
```

Figura 60. Función para escribir en TFRecords.

Esto se hace para cada texto *Review* y valor de *Label* mediante un bucle. Una vez realizado con todos los textos y etiquetas, se escribe un fichero TFRecord tanto para entrenamiento como para test.

```
# Función de tokenización y escritura en TFRecord
def encode(review, label):
    tokens = tokenizer(
        review,  # String
        padding="max_length",
        truncation=True,
        return_tensors="np"
    )
    return tokens["input_ids"][0], tokens["attention_mask"][0], label
```

Figura 61. Función "encode" para tokenización

Posteriormente, si se aprecia la Figura 62, se crea una función para poder leer esos TFRecords almacenados, de manera que define cómo se deben de leer y también decodificar. Esto devuelve un diccionario con las entradas *input_ids, attention_mask* y *label* (Etiquetas).

```
# Función para poder leer los TFRecord creados
def parse_tfrecord_fn(dataRecord):
    feature_description = {
        "input_ids": tf.io.FixedLenFeature([tokenizer.model_max_length], tf.int64),
        "attention_mask": tf.io.FixedLenFeature([tokenizer.model_max_length], tf.int64),
        "label": tf.io.FixedLenFeature([], tf.int64),
    }
    dataRecord = tf.io.parse_single_example(dataRecord, feature_description)
    return {
        "input_ids": dataRecord["input_ids"],
        "attention_mask": dataRecord["attention_mask"]
    }, dataRecord["label"]
```

Figura 62. Función para escribir en TFRecords

Esta función es llamada posteriormente, ilustrándose en la Figura 63, para crear los conjuntos de datos de entrenamiento y de test. En este punto se convierten los ficheros *TFRecords* en objetos de tipo *tf.data.Dataset.* Esto permite poder utilizar unidades TPU, que son más eficientes en los entrenamientos. En el caso de la versión gratuita del entorno, existe la posibilidad de utilizar la TPU, dependiendo de los momentos de baja carga de los servidores de Google. De forma que se realiza esta transformación para intentar aprovechar esta ventaja cuando sea posible, aunque no siempre se puede obtener este beneficio ya que depende mucho cómo se ha comentado, de qué haya baja carga en los servidores.

```
# Leemos los TFRecords
train_dataset = tf.data.TFRecordDataset("train.tfrecord").map(parse_tfrecord_fn).batch(8).prefetch(tf.data.AUTOTUNE)
test_dataset = tf.data.TFRecordDataset("test.tfrecord").map(parse_tfrecord_fn).batch(8).prefetch(tf.data.AUTOTUNE)
```

Figura 63. Lectura de los ficheros TFRecords

Para terminar, se define el modelo que se va a utilizar en el entrenamiento que coincide con el nombre del tokenizador, definido previamente con un número de etiquetas definido en la variable *num_labels*. Cómo se ha comentado se trata de modelos preentrenados, por lo tanto, se congelan los pesos de las capas ocultas, para ajustar los pesos de las capas de salida, qué es una técnica conocida como *fine-tuning*. En la se aprecia en la definición del modelo, ya que se toma con una función que indica que está preentrenado (*from_pretrained*).

```
# Cargamos el modelo preentrenado, debe coincidir con el tokenizador
model_name = model_token_name
num_labels = len(np.unique(df_train["Label_Encoded"]))

# Definición, compilación y entrenamiento del modelo
huggingFace_model = TFAutoModelForSequenceClassification.from_pretrained(model_name, num_labels=num_labels)
huggingFace_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=["accuracy"]
)
history = huggingFace_model.fit(
    train_dataset,
    validation_data=test_dataset,
    epochs=10,
    verbose=2
)
```

Figura 64. Definición del modelo y entrenamiento

Posteriormente, se compila el modelo y se ajusta, pero no existe diferencias significativas respecto al caso del entrenamiento de la red neuronal de clasificación que se ha realizado anteriormente.

Una vez definida la estructura que va a permitir manejar el flujo de código de manera eficiente, se pasa a comentar algunas de las arquitecturas utilizadas. En este punto, es importante destacar que se han usado los modelos más pequeños encontrados, puesto que, para este entorno limitado, si se intenta ejecutar un modelo mediano o grande, simplemente es imposible por la gestión de la memoria RAM.

TinyBERT

El primer modelo usado para el entrenamiento de los datos, es TinyBERT. Se trata de una versión comprimida de BERT (Bidirectional Encoder Representations from Transformers), creada para ser más eficiente en uso de recursos computacionales sin comprometer en exceso el rendimiento de tareas en el procesamiento de lenguaje natural, como comenta Durgia C. (2021). Algunas de las principales virtudes de este modelo, es que se trata de un modelo compacto ya que comparado con un modelo BERT, utiliza menos parámetros lo que lo hace más ligero para dispositivos menos potentes. Sí que tiene la capacidad de mantener un alto rendimiento a pesar de esta reducción de parámetros, aunque evidentemente no es tan potente como un modelo BERT completo. Y también es eficiente, ya que consigue ser rápido a la vez que consume menos memoria que un modelo BERT estándar, y esto lo hace ideal para dispositivos o entornos de trabajo con recursos limitados como es este caso. Para hacer uso de él, se debe colocar en el nombre del tokenizador, el nombre del modelo y empezar a entrenar. En este caso se coloca en la variable *model_token_name* el nombre "*huawei-noah/TinyBERT_General_4L_312D"* y se comienza a entrenar.

Para el entrenamiento, la tasa de aprendizaje se ha mantenido en 0.0001 qué ha sido la menor tasa de aprendizaje qué se ha podido conseguir, y que aun así se trata de una tasa de aprendizaje suficientemente pequeña como para poder afinar bastante el modelo. En las lecturas de los ficheros TFRecords, el tamaño del batch es de 16 siendo este el máximo que se ha podido conseguir, recordando que el batch es el tamaño del subconjunto de datos que se procesará de manera conjunta en una sola pasada a través del modelo durante el entrenamiento.

Cabe destacar, que el entrenamiento en este caso ha durado en torno a unos 45 o 50 minutos haciendo uso de la GPU que se posee en el entorno gratuito, aunque bien es cierto que no siempre está disponible ya que depende de lo liberado que estén los servidores De Google. De todas formas, el rendimiento en CPU es parejo por lo que no llega a agotar la sesión por inactividad. Las sesiones por inactividad están en torno a la hora u hora y media, por lo que en este caso ha sido suficiente para poder completar el entrenamiento sin mayores problemas. Aquí se pone de manifiesto una de las características del modelo y es que es eficiente dada la muy pequeña cantidad de recursos de la que se dispone. Finalmente comentar que el entrenamiento se ha realizado con una configuración de 10 épocas, hay que recordar que es el número de veces que el modelo consume los datos por completo durante el entrenamiento.

Una vez terminada la construcción del flujo de trabajo, es momento de analizar los resultados obtenidos. En primer lugar, se consigue una Accuracy del 92% lo que indica que el modelo clasifica correctamente el 92% de las muestras en general, siendo un resultado excelente. Por clases, la precisión de la clase 1 es la más baja siendo esta 0.89, la de la clase 2, 0.91 y la de la clase 0, 0.94. Los resultados son simplemente geniales para cada una de las clases siendo en el peor de los casos para la clase 1 qué es la que ha conseguido la puntuación más baja. En cuanto al Recall, se tiene algo parecido ya que para la clase 1 sigue siendo el más bajo con una puntuación de 0.88, mientras que en la clase 0 y clase 2 se obtienen 0.92 y 0.95 respectivamente. En general el modelo identifica los ejemplos de cada clase de manera correcta siendo en el mejor

de los casos para la clase 2 en el que apenas se equivoca. Finalmente, la característica F1-Score que de nuevo la puntuación más baja obtenidas para la clase 1 con 0.89, y siendo superior en las otras dos clases con una puntuación de 0.93. Esto refleja un buen balanceo entre clases para cada una de las 3 clases existentes, aunque en el caso de la clase 1, es el que tiene un poco más de dificultades.

```
Resultados:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93      5856
           1       0.89      0.88      0.89      5221
           2       0.91      0.95      0.93      6447

    accuracy                           0.92     17524
   macro avg       0.92      0.92      0.92     17524
weighted avg       0.92      0.92      0.92     17524

Matriz de Confusión:
[[5377  305  174]
 [ 218 4610  393]
 [  96  252 6099]]
AUC-ROC Score: 0.98
```

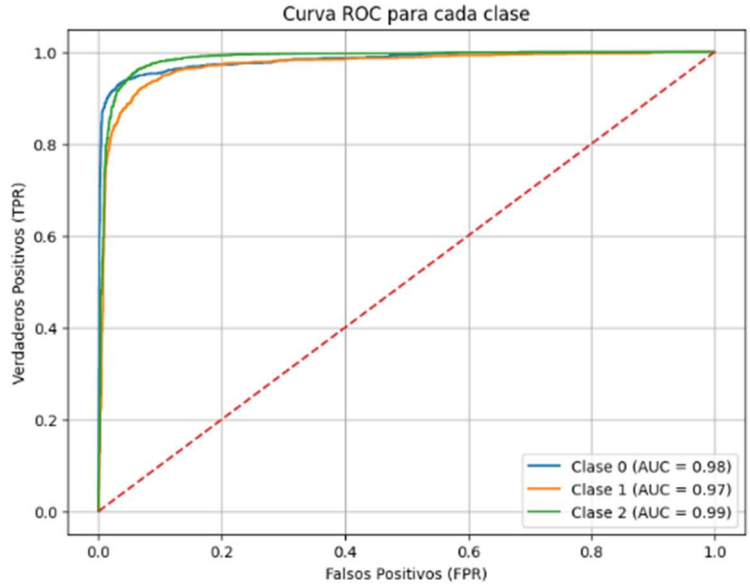


Figura 65. Resultados obtenidos para TinyBERT

En cuanto a la matriz de confusión, se muestra el rendimiento de cada una de las clases y se aprecia que en la diagonal principal que es donde se contienen los verdaderos positivos se obtienen números muy elevados manifestando la gran capacidad que tiene el modelo para identificar correctamente cada una de las clases. Sin embargo, fuera de esta diagonal principal, se encuentra los errores a la hora de predecir cada clase. En este sentido se puede apreciar cómo la clase 1 es la que detecta más errores como se ha comentado anteriormente.

En cuanto a las curvas, se puede observar cómo gráficamente se obtiene que la puntuación global del área bajo la curva es de 0.98 algo excelente que, si se pasa a analizar por clases, la menor puntuación obtenida es para la clase 1, con una puntuación de área bajo la curva de 0.97, y un AUC de prácticamente 1 para la clase dos. Como conclusión general este modelo muestra un rendimiento muy bueno dadas las puntuaciones obtenidas en cada una de las métricas.

DistilBERT

En este caso se trata también de una versión reducida de BERT, diseñada por HuggingFace para ser más ligera y eficiente en términos computacionales mientras mantiene un rendimiento potente para tareas como NLP como comenta Malingan N. (2024). Por tanto, se trata de un

modelo más comprimido que BERT teniendo una reducción de aproximadamente un 35% de parámetros respecto a BERT, manteniendo aproximadamente el rendimiento con respecto a este, y la reducción de parámetros permite realizar entrenamientos más rápido por lo que para dispositivos con rendimiento limitado es un modelo muy interesante. Para hacer uso de él, lo único que se debe de hacer es cambiar en el código el nombre del tokenizador, que coincide también con el nombre del modelo, y es "distilbert-base-uncased". En este caso el tiempo de entrenamiento se alargó hasta la hora y 10 minutos, y en este caso permitió completar el entrenamiento sin mayores problemas, aunque siempre estando pendientes de qué no se agotase la sesión por inactividad ya que, al ejecutar una celda en la versión gratuita del entorno, el tiempo por inactividad puede oscilar entre una hora u hora y media y en ese momento se puede terminar la sesión por inactividad. En este caso no fue así y permitió obtener resultados.

Cómo parámetros destacables, la tasa de aprendizaje se mantiene en 0.0001 al igual que en el modelo anterior, y de la misma manera el tamaño de batch se mantiene en 8, para ahorro de recursos. En este caso no se ha variado nada más que el nombre del modelo y tokenizador.

En cuanto a las métricas, en la Figura 66 se puede apreciar que en Precisión se consigue para la clase 1, 0.59, para la clase 2, 0.72 y para la clase 0, 0.71. Esto indica que la clase 1 tiene más falsos positivos en comparación a las otras dos clases, pero el rendimiento en general comparado con el modelo anterior es bastante menor. En cuanto al Recall, la clase 0 obtiene 0.55, la clase 1 obtiene 0.72 y la clase dos obtiene 0.73. No siendo unos valores tan buenos como en el modelo anterior, la clase 1 y la clase 2 tienen un Recall bueno lo que implica que este modelo identifica correctamente la mayoría de los ejemplos de estas dos clases y sin embargo para la clase restante es bastante peor por lo que los ejemplos de esta clase 0, no se identifican correctamente. Finalmente, en cuanto a F1-Score, para la clase 0 se tiene una puntuación de 0.62, para la clase 1 una puntuación de 0.65 y para la clase 2 una puntuación de 0.72. El mejor equilibrio por tanto entre precisión y Recall recae sobre la clase dos, y es el caso de las otras dos clases donde se tiene mayor problema de equilibrio entre estas dos métricas.

La Accuracy, indica que el modelo clasifica el 67% de las muestras en general correctamente, se trata de un rendimiento aceptable, pero tiene bastante margen de mejora. En cuanto a la matriz de confusión se puede apreciar que las confusiones más grandes se producen entre las clases adyacentes, es decir hay mucha confusión entre las clases 1 y 2, y entre las clases 0 y 1. En cuanto a la puntuación del área bajo la curva global es de 0.84, qué es un valor bueno o razonablemente bueno para que el modelo distinga entre clases. Por clases la puntuación más baja, es para la clase 1, mientras qué es ligeramente superior en la clase 2.

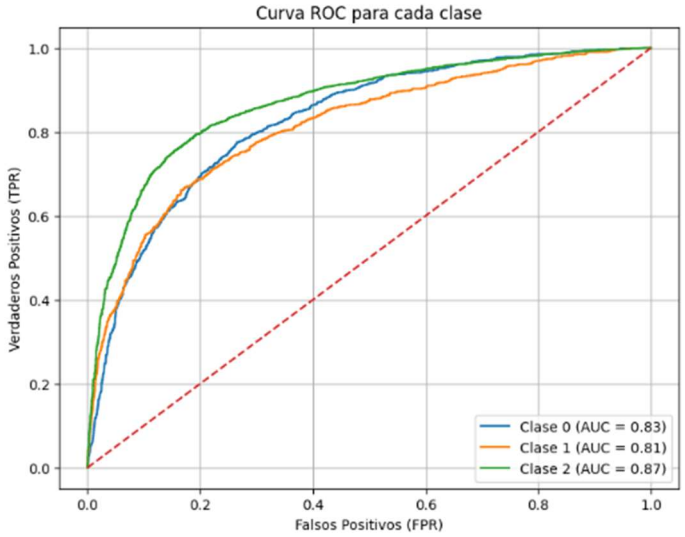
```
Resultados:
              precision    recall  f1-score   support

           0       0.71      0.55      0.62      6380
           1       0.59      0.72      0.65      6210
           2       0.72      0.73      0.73      6210

    accuracy                           0.67     18800
   macro avg       0.68      0.67      0.67     18800
weighted avg       0.68      0.67      0.67     18800


Matriz de Confusión:
[[3522 1985  873]
 [ 838 4491  881]
 [ 591 1088 4531]]

AUC-ROC Score: 0.84
```




```
Tiempo de ejecución total: 4686.03 segundos
```

Figura 66. Resultados del modelo DistilBERT

Otros modelos

En la búsqueda por probar nuevos modelos y poder realizar más comparativa de resultados para seguir estudiando las posibilidades de más modelos, se siguió buscando modelos de rendimiento potente pero que fuesen livianos en cuanto a la utilización de recursos. Algunos de ellos son MiniLM qué es de Microsoft o MobileBERT de Google, pero a la hora de realizar los entrenamientos los tiempos de espera entre épocas eran enormes, de forma que se llegaba no ha consumir los recursos de RAM, pero sí que se llegaba a consumir la sesión por inactividad.

En la Figura 67, se puede apreciar cómo por ejemplo en el modelo MobileBERT de Google, el tiempo para completar una época fue de aproximadamente 50 minutos, y durante el transcurso de la segunda época la sesión fue terminada por inactividad ya que, aunque se había ejecutado esta celda, si la sesión detecta que no se ejecuta en este tiempo una nueva celda cancela la sesión por inactividad.

Por tanto, estos modelos no eran tan eficientes computacionalmente como los dos anteriores que sí ha permitido poder realizar un entrenamiento lo suficientemente rápido o en un periodo de tiempo aceptable, para que en este entorno se pudieran obtener resultados. Por tanto, no se ha podido obtener resultados para más modelos debido a las limitaciones del entorno existentes, aunque sí se ha podido explorar al menos con dos modelos cómo es el funcionamiento de la librería Transformers de HuggingFace.

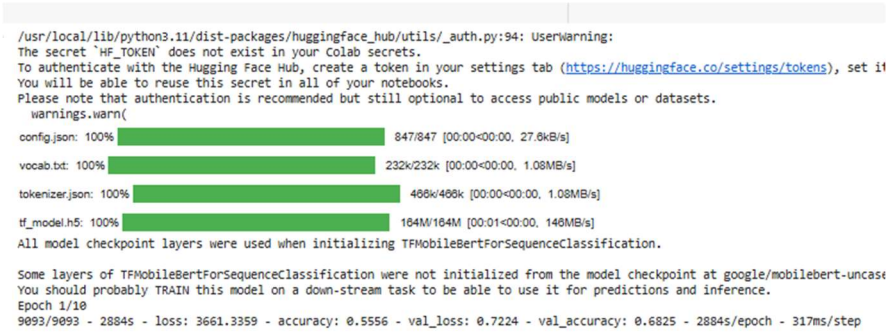


Figura 67. Tiempo de una época en MobileBERT

Selección del modelo

Para finalizar este estudio, se debe elegir uno de los modelos que se ha estudiado durante el proyecto para concluir cuál sería el mejor de acuerdo a los datos que se disponen. De los modelos analizados, se tiene que Random Forest tiene un rendimiento parecido a XGBoost pero ligeramente menor en precisión y F1, además tiene problemas en la clase cero.

En cuanto a XGBoost, tiene un buen balance entre clases, aunque es más bajo que en el caso de TinyBERT en Recall y precisión para algunas clases. La red neuronal tiene un buen rendimiento general para las clases involucradas, sin embargo, el área bajo la curva es menor que en los modelos de árboles.

En TinyBERT, tiene un gran balance en todas las métricas y clasifica prácticamente sin errores en todas las clases, aunque es un modelo más complejo que los anteriores. La gran mayoría de las métricas obtenidas, tienen una puntuación superior a 0.90, lo que leas el modelo más potente de todos.

Finalmente está DistilBERT, que tiene un rendimiento notablemente más bajo en la mayoría de métricas y en comparación a otros modelos incluso más simples es menos competitivo. En el caso de los modelos como Regresión Logística o Naive Bayes, el rendimiento es paupérrimo comparado con los otros modelos, por lo que no se tienen en consideración para la elección.

Para centrarse en la elección, en la Tabla 2 se resumen las puntuaciones obtenidas por cada modelo, clase por clase, media de clases y media global de las puntuaciones como los casos de Accuracy o AUC. Recordar de nuevo en este punto, que la clase 0 se corresponde con la etiqueta “negative” (reseña negativa), la clase 1 se corresponde con la etiqueta “neutral” (reseña neutra) y la clase 2 se corresponde con la etiqueta “positive” (reseña positiva).

	Clase/ Global	Precisión	Recall	F1-Score	AUC-ROC	Accuracy
Regresión Logística	**0**	0.26	0.17	0.20	0.49	0.29
	1	0.26	0.38	0.30		
	2	0.35	0.37	0.36		
	Media	0.29	0.30	0.29		
Naive Bayes Gaussiano	**0**	0.24	0.11	0.15	0.55	0.37
	1	0.40	0.51	0.45		
	2	0.41	0.60	0.48		
	Media	0.35	0.41	0.36		
DistilBERT (HuggingFace)	**0**	0.71	0.55	0.62	0.84	0.67
	1	0.59	0.72	0.65		
	2	0.72	0.73	0.73		
	Media	0.68	0.67	0.67		
Random Forest	**0**	0.96	0.65	0.78	0.98	0.81
	1	0.61	0.98	0.75		
	2	1.00	0.85	0.92		
	Media	0.85	0.83	0.81		
Red Neuronal (8 capas, hasta 256 neuronas)	**0**	0.94	0.78	0.85	0.94	0.82
	1	0.73	0.96	0.83		
	2	0.80	0.76	0.78		
	Media	0.82	0.83	0.82		
Gradient Boosting (XGBoost)	**0**	0.97	0.67	0.79	0.98	0.83
	1	0.63	0.98	0.77		
	2	1.00	0.91	0.95		
	Media	0.87	0.85	0.84		
TinyBERT (HuggingFace)	**0**	0.94	0.92	0.93	0.98	0.92
	1	0.89	0.88	0.89		
	2	0.91	0.95	0.93		
	Media	0.92	0.92	0.92		

Tabla 2. Comparativa de puntuaciones de las métricas de los modelos estudiados

Atendiendo a la Tabla 2, teniendo en cuenta las limitaciones del entorno existentes y para el conjunto de datos que se dispone para realizar este estudio o proyecto, el modelo que mejor se comporta para estos datos es TinyBERT que aun siendo el más complejo de todos, es el que arroja mejores resultados en todas las áreas de puntuación, por lo que le hace el modelo idóneo para la clasificación de las reseñas de los productos de alimentación de Amazon, para determinar el sentimiento de las reseñas.

Conclusiones

A lo largo de este proyecto se ha comprobado la complejidad que tiene un estudio de este tipo, así como los distintos problemas que pueden ir surgiendo a lo largo del estudio, y cómo se deben de ir sorteando mediante distintas alternativas y toma de decisiones justificada en base a los datos que se van obteniendo en las distintas transformaciones. También se ha comprobado la complejidad que tiene un proyecto de esta envergadura, al disponer de un entorno con recursos

limitados, lo que acota la toma de decisiones ya que se deben de tener en cuenta estas limitaciones a lo largo del desarrollo.

Con esto, cobra especial importancia la capacidad de adaptación a las distintas problemáticas que van sucediéndose a lo largo de un proyecto de Data Science, y las implicaciones que puede conllevar cada decisión. Estas siempre deben ser justificadas y documentadas debidamente con el fin de obtener un estudio lo más riguroso posible.

Finalmente, se ha comprobado el potencial que ofrece la ciencia de datos y las múltiples aplicaciones que puede tener en infinidad de campos. En este caso, el aprendizaje de lenguaje natural ofrece posibilidades muy potentes, de cara a aplicaciones muy variadas y que aun pueden estar por descubrir muchas otras finalidades.

Bibliografía

- Kaggle. (n.d.). *Amazon Fine Food Reviews*. Kaggle Datasets. Recuperado de https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews
- Pereira D. (2024) Modelo de negocio de Amazon. Recuperado en 10 de diciembre de 2024, de: https://businessmodelanalyst.com/es/modelo-de-negocio-de-amazonas/?srsltid=AfmBOopTPlpALC19S9vuSzdyLbLlmYk0lbCUPvIynzEgk21PNrb5ZLFI
- Beucher M. (2024). Descubre el modelo de negocios de Amazon que lo llevó a ser el número 1 en el E-commerce. Recuperado el 10 de diciembre de 2024, de: https://espacioempresa.com/lideres/modelo-negocios-amazon-estrategias/
- Conroy M. (2023) CRISP-DM, Phase 3: Data Preparation. Recuperado el 14 de diciembre de 2024, de: https://datalabnotes.com/data-preparation-crisp-dm/
- Carney S. (2020). CRISP-DM in Depth: data preparation. Recuperado el 15 de diciembre de 2024, de: https://www.seancarney.ca/2020/10/10/crisp-dm-in-depth-data-preparation/
- Elgort B. (2022) Using the Python Inflect Library. Recuperado el 22 de diciembre de 2024, de: https://bruceelgort.com/2022/12/23/using-the-python-inflect-library/
- Mielse P. (2023) What is Cosine Similarity: A comprehensive Guide. Recuperado el 25 de diciembre de 2024, de: https://www.datastax.com/guides/what-is-cosine-similarity
- Fiallos J. (2023) Similitud de coseno y Jaccard: Midiendo semejanzas en Python. Recuperado el 25 de diciembre de 2024, de: https://jackfiallos.com/similitud-coseno-y-jaccard-midiendo-semejanzas-en-python
- Auwal Khalid A. (2023) Textblob: A Simple Library forNatural Language Processing (NLP). Recuperado el 1 de enero de 2025, de: https://medium.com/@khalidassalafy/textblob-a-simple-library-for-natural-language-processing-nlp-81b993e252b0
- Ersan Yagci H. (2021). Under-Sampling Methods for Imbalanced Data (ClusterCentroids, RandomUnderSampler, NearMiss). Recuperado el 2 de enero de 2025, de: https://hersanyagci.medium.com/under-sampling-methods-for-imbalanced-data-clustercentroids-randomundersampler-nearmiss-eae0eadcc145
- Brownlee J. (2021). SMOTE for Imbalanced Classification with Python. Recuperado el 9 de enero de 2025, de: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
- Verma Y. (2021) Beginners Guider To Truncated SVD For Dimensionality Reduction, recuperado el 10 de enero de 2025, de https://analyticsindiamag.com/deep-tech/beginners-guide-to-truncated-svd-for-dimensionality-reduction/
- Navarro S. (2024). ¿Qué es HuggingFace?. Recuperado el 10 de diciembre de 2024, de: https://keepcoding.io/blog/que-es-hugging-face/
- López F. (2019) Construcción de un modelo de machine Learning usando Tensorflow (parte 2). Recuperado el 13 de enero de 2025, de: https://evenbytes.com/es/construccion-de-un-modelo-de-machine-learning-usando-tensorflow-parte-2/
- Dimitre Oliveira (2023). Creating TFRecords. Recuperado el 15 de enero de 2025, de: https://keras.io/examples/keras_recipes/creating_tfrecords/
- Durgia C. (2021) Exploring BERT variants (Part 2): SpanBert, DistilBERT, TinyBERT. Recuperado el 20 de enero de 2025, de: https://towardsdatascience.com/exploring-bert-variants-part-2-spanbert-distilbert-tinybert-8e9bbef4eef1
- Malingan N. (2024) HugginFace DistilBERT. Recuperado el 20 de enero de 2025, de: https://www.scaler.com/topics/nlp/distilbert/
- Efimov V. (2023) Large Language Models, ALBERT – A Lite BERT for Self-supervised Learning, recuperado el 22 de enero de 2025, de: https://towardsdatascience.com/albert-22983090d062

- Tsang S. (2022) Review – MobileBERT: A Compact Tast-Agnostic BERT for Resource-Limited Devices. Recuperado el 22 de enero de 2025, de: https://sh-tsang.medium.com/review-mobilebert-a-compact-task-agnostic-bert-for-resource-limited-devices-b73078d335df

Anexo

Índice de Tablas

Índice de Figuras