PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

ANÁLISIS DE SENTIMIENTO EN LA INDUSTRIA DEL LUJO APLICANDO PROCESAMIENTO DE LENGUAJE NATURAL (NLP) Y MACHINE LEARNING

Tesis para obtener el título profesional de Ingeniero Mecatrónico

AUTOR:

JEFFERSON ERICK OSORIO ESPINOZA

ASESOR:

ELIZABETH ROXANA VILLOTA CERNA, Ph.D.

Lima, febrero, 2025

Informe de Similitud

Yo, Elizabeth Roxana Villota Cerna, docente de la Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú, asesor(a) de la tesis/el trabajo de investigación titulado ANÁLISIS DE SENTIMIENTO EN LA INDUSTRIA DEL LUJO APLICANDO PROCESAMIENTO DE LENGUAJE NATURAL (NLP) Y MACHINE LEARNING, del autor Jefferson Osorio, dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 14%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 27/02/2025.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: 27/02/2025

Apellidos y nombres de la asesora: Villota Cerna, Elizabeth Roxana	
DNI: 10686413	Firma 
ORCID: 0000-0002-6431-1479	

RESUMEN

Esta tesis se enfoca en el análisis avanzado de sentimiento aplicando técnicas de procesamiento de lenguaje natural (NLP) para examinar las evaluaciones y comentarios de clientes en el ámbito del lujo. A través de un proceso que incluye la recopilación y preparación de los datos, modelado de NLP y *machine learning*, así como la implementación y análisis de resultados, se busca mejorar la comprensión de la percepción de la marca y la satisfacción del cliente. El objetivo de esta tesis es brindar a las empresas de lujo conocimientos prácticos para mejorar sus productos con la finalidad de optimizar la experiencia del cliente y, de esta manera, fortalecer su posición en el ámbito del lujo.

En el primer capítulo se presenta la introducción al análisis de datos, abordando técnicas generales de análisis y describiendo modelos de *machine learning* y *deep learning*, enfatizando su importancia en la extracción de información a partir de conjuntos de datos que incluyen reseñas de productos por parte de los consumidores. Así mismo, se detalla la problemática a resolver y los objetivos planteados para el desarrollo de la presente tesis

En el segundo capítulo se desarrolla el marco teórico y estado del arte. En el marco teórico se da una breve explicación de las métricas que ayudan a seleccionar al algoritmo más adecuado para realizar un análisis de sentimiento, así como también la explicación de su proceso de implementación. Por otro lado, en el estado del arte se detallan hitos asociados al NLP que han contribuido a que este tipo de análisis cada vez sea más preciso.

La estrategia para realizar un preprocesado de datos e implementar un análisis exploratorio se desarrolla en el tercer capítulo. Aquí se describe el conjunto de datos a emplear, se describen las librerías más usadas para realizar un análisis exploratorio en el marco de desarrollar un análisis de sentimiento lo que permite que el conjunto de datos este preparado para implementar los algoritmos o modelos de NLP.

En el cuarto capítulo y, considerando el análisis exploratorio previo, se desarrolla la función que se encarga de procesar las reseñas o comentarios de los consumidores hacia los productos adquiridos. Se estudian además los *N-gramas* para entender a detalle el conjunto de datos y, de la misma manera, conocer qué tan bien la función implementada cumple su función de procesar estos textos para realizar el modelado posterior.

En el capítulo quinto se desarrollan los modelos de *machine learning* y *deep learning* para implementar el análisis de sentimiento. Así mismo, se selecciona el mejor modelo, haciendo una comparativa de métricas, para ser implementado como el modelo que da mejores resultados al momento de predecir cuándo una reseña es positiva o negativa.

En el capítulo sexto se pone a prueba el modelo seleccionado a través de un nuevo conjunto de datos para ver cuán preciso es este modelo frente a nuevas reseñas de los consumidores. Esto se desarrolla a través de métricas para evaluar al modelo y concluir que es la mejor opción en este contexto en particular.

Por último, el análisis de costo de un proyecto de análisis de sentimiento y las conclusiones de la tesis se desarrolla en el séptimo y octavo capítulo respectivamente. Además, se enfatizan las lecciones aprendidas y se ofrecen recomendaciones para futuras investigaciones o aplicaciones prácticas de las técnicas de NLP en el análisis de opiniones o reseñas.

DEDICATORIA

A mi madre, quien ha sido y siempre será mi mayor fuente de inspiración. Este trabajo es un tributo a tu amor, tu fortaleza y tu fe inquebrantable en mí. Desde siempre impulsaste mis sueños con pasión, creyendo en mi capacidad incluso cuando yo mismo dudaba. Este logro lleva tu esencia, porque fueron tus palabras alentadoras y tu ejemplo de vida los que me dieron la fuerza para superar cada desafío en este camino.

Aunque físicamente ya no estás presente, sé que este momento te llenaría de orgullo y felicidad, porque cumplir este sueño era también parte de los tuyos. Puedo imaginar tu sonrisa, tu mirada llena de emoción, y escuchar esas palabras que tantas veces me repetiste: "Continua adelante, tú puedes lograrlo". Este trabajo no habría sido posible sin el amor y la energía que sembraste en mi corazón. Gracias, Mamita Gladys, por guiarme siempre. Este trabajo es para ti, con todo el amor que te debo y que siempre te guardaré.

A mis hermanos, Sandra y Patrick, un ejemplo constante de perseverancia, unión y amor fraternal. Gracias por estar a mi lado, por su apoyo y las palabras de ánimo que nunca faltaron en este largo proceso. Ustedes me han enseñado que, incluso frente a las adversidades, la familia es la mayor fortaleza. Agradezco, además, la dicha de tener a sus maravillosos hijos, mis sobrinos, quienes han llenado nuestras vidas de alegría y han sido un recordatorio constante de que lo más importante es la familia.

AGRADECIMIENTO

Quiero agradecer a la Dra. Elizabeth Villota Cerna, mi asesora de tesis, por todo el apoyo que me ha brindado a lo largo de este proceso. Más allá de su orientación académica, su compromiso y paciencia han sido elementos esenciales para desarrollar el presente trabajo. Gracias a su guía, he podido avanzar con confianza, sintiéndome acompañado en cada fase del proyecto.

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1 – INTRODUCCIÓN

En la actualidad, analizar grandes conjuntos de datos y, sobre todo, textuales se ha convertido fundamental para múltiples industrias. El procesamiento de lenguaje natural (NLP) se ha destacado como una tecnología fundamental para extraer información relevante de estos datos, permitiendo a las empresas comprender mejor las reseñas que realizan sus consumidores a los productos adquiridos por ellos. Esto es especialmente significativo en el sector del lujo, donde las percepciones y sentimientos de los consumidores influyen considerablemente en las estrategias, sobre todo, de marketing, la mejora de la experiencia del cliente y la preservación de la exclusividad y prestigio de las marcas.

El sector del lujo se distingue por ofrecer productos y servicios de alta calidad que buscan proporcionar experiencias únicas y personalizadas. Sin embargo, la naturaleza subjetiva y compleja del lenguaje humano presenta desafíos importantes para realizar un análisis de sentimiento más preciso y automatizado. Las marcas de lujo requieren herramientas avanzadas capaces de interpretar correctamente las emociones expresadas por sus consumidores a través de las reseñas de productos.

Esta tesis propone un enfoque integral para desarrollar e implementar modelos de *machine learning* destinados al análisis de sentimientos. Se evalúan y comparan diversos algoritmos para seleccionar al más efectivo. Además, se centra específicamente en el sector del luego, para ofrecer una solución que permita a las marcas obtener *insights* detallados sobre cómo sus consumidores perciben sus productos. Así mismo, la presente tesis está diseñada para equipar a las marcas de lujo con las herramientas necesarias para interpretar las opiniones y emociones de los consumidores, facilitando la adaptación de sus estrategias comerciales. Todas las decisiones están orientadas a beneficiar tanto a los consumidores como a los objetivos comerciales de las marcas, asegurando la alineación entre las estrategias implementadas y las percepciones del público objetivo.

1.1 Presentación de la problemática

Dentro del ámbito del lujo, las marcas enfrentan el desafío constante de mantener un prestigio y cumplir con las expectativas cada vez más exigentes de sus clientes. Una herramienta para mejorar en este aspecto es el análisis de sentimientos de reseñas y valoraciones de productos realizadas por los consumidores, que proporciona *insights* sobre las percepciones y emociones de estos últimos. Marcas como Louiss Vuitton y Chanel, por ejemplo, utilizan el análisis de sentimiento para identificar tendencias emergentes y ajustar sus estrategias de mercado y comunicación lo que, al largo plazo, les trae beneficios. Sin embargo, el análisis efectivo de estos datos requiere un enfoque capaz de manejar la complejidad y sutileza de los textos relacionados con el lujo, donde las expresiones de satisfacción o insatisfacción no siempre son directas o explícitas. Las emociones subyacentes en estos textos son variadas y multifacéticas, lo que demanda un método que no solo identifique palabras clave, sino que también comprenda los matices implícitos en ellas. En esta tesis se detalla cómo, mediante el uso de técnicas de procesamiento de lenguaje natural (NLP), es posible captar estos matices de manera efectiva.

1.2 Objetivos de la tesis

El objetivo principal de esta tesis es desarrollar un sistema de análisis de sentimientos utilizando técnicas de procesamiento de lenguaje natural (NL) y *machine learning*, para categorizar las reseñas y evaluaciones de los consumidores en el sector del lujo. A través de este análisis, se busca entender la dinámica del sentimiento del consumidor y ajustar estrategias de productos y comunicación que no solo satisfagan, si no que superen las expectativas de los consumidores. De esta manera, se busca fortalecer la fidelidad del cliente y maximizar el valor de la marca.

1.2.1 Objetivos específicos de la tesis

a. Realizar una exploración inicial de las reseñas y evaluaciones de los consumidores para identificar aspectos clave del conjunto de datos, como la distribución de sentimientos, las palabras más utilizadas y posibles sesgos presentes en los comentarios.
b. Crear un modelo de análisis de sentimiento utilizando técnicas de procesamiento de lengua natural (NLP), para analizar de manera profunda las reseñas de los consumidores, con un enfoque específico en el ámbito empresarial.
c. Implementar algoritmos de aprendizaje automático (*machine learning*) para clasificar los sentimientos de los clientes en positivos y negativos, priorizando la identificación de opiniones que puedan influir en las decisiones estratégicas de la compañía, sin considerar las categorías neutrales.
d. Identificar patrones y tendencias en las opiniones de los consumidores que influyan en la imagen de marca y el posicionamiento de la marca, mediante la extracción relevante a partir de los datos textuales.
e. Comparar las métricas, por ende, el rendimiento, de diferentes técnicas de *machine learning* (incluyendo enfoques basados en modelos supervisados y redes neuronales) para determinar cuál ofrece los mejores resultados en este contexto y seleccionar al mejor modelo en este contexto.
f. Proponer recomendaciones prácticas para las marcas de lujo sobre cómo utilizar los hallazgos del análisis de sentimientos para mejorar la experiencia del cliente.

1.3 Alcance y limitaciones de la tesis

El alcance de esta tesis es desarrollar un sistema de análisis de sentimientos para el sector de lujo utilizando técnicas de procesamiento de lenguaje natural (NLP) y *machine learning*. Se emplean tecnologías accesibles y gratuitas como *Google Collab* para la programación y experimentación, lo cual facilita el acceso a recursos computacionales. Esta elección es adecuada para el entorno inicial de investigación y desarrollo del estudio. No obstante, se reconoce que para implementaciones empresariales se requerirían recursos más potentes y especializados para manejar grandes volúmenes de datos con la velocidad y eficiencia necesaria.

Entre las limitaciones de este estudio se encuentra la dificultad para capturar matices sutiles en las reseñas relacionadas con el sector del lujo, lo que podría afectar la precisión del análisis de sentimientos. Además, la capacidad para generalizar los resultados obtenidos a partir de una muestra específica de datos podría limitar su aplicabilidad a toda la industria del lujo. Cabe resaltar que este tipo de análisis son diseñados especialmente para una marca en concreto ya que el conjunto de datos varía de una marca a otra por lo que no podría aplicarse de manera general. Sin embargo, se desarrolla el procedimiento esencial a seguir para implementar esta solución.

1.4 Descripción de la propuesta de solución

El desarrollo de este sistema comienza con la identificación y extracción de datos. En este caso, la empresa *Luxury Beauty* proporcionó los datos necesarios para realizar un estudio de análisis de sentimiento considerando estos datos accesibles para cualquier persona que quiera implementar un análisis a partir de su conjunto de datos. Es relevante destacar que la extracción de datos se realiza a través de plataformas en línea, que incluyen redes sociales, foros y sitios donde la marca y sus productos tienen presencia.

La preparación de los datos implica técnicas de limpieza como la eliminación de duplicados, corrección de errores tipográficos y normalización del texto para asegurar la coherencia del conjunto de datos. Posteriormente, se emplean modelos de procesamiento de lenguaje natural (NLP) diseñados para captar la complejidad del lenguaje utilizado en las reseñas de productos. Estos modelos son entrenados con el conjunto de datos proporcionados por la empresa.

Para interpretar los resultados de manera efectiva, se genera una tabla resumen y se realizan visualizaciones comparativas entre los resultados obtenidos por cada modelo de *machine learning* o *deep learning* empleado.

CAPÍTULO 2 – MARCO TEÓRICO Y ESTADO DEL ARTE

En este capítulo se presenta una revisión del marco teórico y del estado del arte en el análisis de sentimientos utilizando procesamiento de lenguaje natural (NLP) centrado específicamente en el sector del lujo. Se abordan conceptos fundamentales de NLP, incluyendo técnicas de preprocesamiento como *tokenización*, *stemming*, lematización y eliminación de *stop words*, además de métodos avanzados de representación de texto como *TF-IDF* (*Term Frequency-Inverse Document Frequency*), *embedding* de palabras (por ejemplo, *Word2Vec* y *GloVe*) y modelos preentrados como *BERT* (*Bidirectional Encoder Representations from Transformers*) y *GPT* (*Generative Pre-trained Transformer*) los cuales serán mencionados como parte de las recomendaciones en la presente tesis.

Se revisan los principales algoritmos *machine learning* utilizados en el análisis de sentimientos, tales como la regresión logística, *Support Vector Machines* (SCV, Máquinas de Vectores de Soporte), *Ramdom Forest* y técnicas de *deep learning* como Redes Neuronales Recurrentes (RNN, *Recurrent Neural Network*), *LSTM* (*Long Short-Term Memory*) y transformadores (*Transformers*).

Se incluyen estudios previos sobre aplicaciones prácticas en el sector del lujo, destacando cómo estas técnicas ayudan a comprender las percepciones y emociones de los consumidores, El capítulo explora los principios fundamentales del NLP, ofreciendo una visión detallada del proceso de selección de herramientas y plataformas, la recopilación y preparación de datos, así como el modelado y entrenamiento de algoritmos.

Además, se describe en detalle la evaluación de modelos de *machine learning*, que incluye métricas como precisión, *recall*, F1-Score, AUC-ROC (Área bajo la curva características y operativa del receptor), la matriz de confusión y su relevancia para interpretar los resultados. También se abordan los desafíos actuales, como la calidad de los datos, el manejo del sarcasmo e ironía, y la necesidad de modelos con buena capacidad de generalización. Así mismo, se exploran las tendencias futuras en NLP, subrayando la importancia de la adaptabilidad y la mejora continua de los modelos para mantener su precisión y relevancia.

2.1 Fundamentos de NLP y análisis de sentimiento

Los principios de NLP se extiende desde la comprensión del lenguaje natural hasta la interpretación de emociones a través del texto. Las tecnologías incluyen modelos de lenguaje capaces de comprender contextos complejos y algoritmos de clasificación que distinguen entre sentimientos positivos, negativos y neutros. Estos modelos se entrenan con conjuntos extensos de datos para detectar patrones y sutilezas propias del sector, así como también permitir una evaluación precisa de las reseñas expresadas hacia los productos.

El uso de bibliotecas de *Python* como *Scikit-Learn* para modelado predictivo y *pandas*, principalmente, para la gestión de datos, junto con técnicas como la vectorización para convertir texto en características numéricas y algoritmos de clasificación como la regresión logística, facilita el desarrollo de sistemas eficaces de análisis de sentimientos. Estos métodos posibilitan la clasificación automática de comentarios en diversas categorías de sentimientos.

2.2 Estado del arte

En este apartado se examinan tanto los desarrollos históricos como los avances más recientes en las técnicas de análisis de datos, centrándose especialmente en cómo la inteligencia artificial (IA) y el NLP han transformado el campo del análisis de sentimientos. Se revisa cómo estas tecnologías han evolucionado desde sus inicios hasta la actualidad, destacando los hitos importantes y las innovaciones que han mejorado la precisión en la interpretación de datos textuales. Se discuten estudios anteriores y aplicaciones prácticas, ofreciendo una visión de cómo estas tecnologías han enriquecido la toma de decisiones.

En los años cincuenta, las investigaciones se enfocaron inicialmente en el desarrollo de sistemas de traducción automática y análisis de estructuras sintácticas, influenciados por los trabajos pioneros de Noam Chomsky. Su obra "*Syntactic Structures*" (1957) introdujo la teoría de la gramática generativa, estableciendo las bases teóricas para desarrollos posteriores. Por otro lado, Minsky (1961) contribuyó con investigaciones sobre el procesamiento semántico, destacando la importancia de representar el significado profundo del lenguaje, fundamental para la comprensión de textos complejos.

En la década de los sesenta, el campo del NLP experimentó avances significativos con la creación de "ELIZA", un programa que simulaba conversaciones humanas utilizando lenguaje natural. Este hito, descrito por Weizenbaum (1986), demostró el potencial de la interacción humado-computadora a través del lenguaje natural. Sin embargo, esta década también vio el informe "ALPAC" (1966) que criticaba el progreso en la traducción automática y marcó el inicio del primer "Invierno de la IA". Este período se caracterizó por una desaceleración en la investigación de la inteligencia artificial debido a expectativas no cumplidas sobre avances tecnológicos.

En los años setenta, la investigación de NLP avanzó más allá de la sintaxis para explorar representaciones semánticas del lenguaje. Fodor (1975) profundizó en el estudio de las gramáticas de casos y las redes semánticas, buscando una representación más profunda del significado de las palabras y sus relaciones dentro de un contexto. Este enfoque ayudo a avanzar a una comprensión más completa del lenguaje, no solo dependiente de estructuras sintácticas.

En los años ochenta, surgieron los sistemas expertos como "MYCIN", que utilizaban reglas predefinidas para el razonamiento en dominios específicos. Estos sistemas, descritos por Shortliffe (1980), demostraron el potencial de la inteligencia artificial aplicada a áreas especializadas como el diagnóstico médico. La aplicación de NLP en estos sistemas proporcionó una plataforma para aplicaciones más complejas que requerían razonamiento especializado.

Los avances en los años noventa marcaron una revolución estadística en el procesamiento de lenguaje, impulsada por el uso de redes neuronales y modelos probabilísticos. La introducción de las redes neuronales recurrentes (RNN) y las redes de memoria a corto y largo plazo (LSTM) permitió a los investigadores modelas las relaciones temporales y secuenciales del lenguaje, mejorando así el análisis de texto y el modelado del lenguaje (Hochreiter & Schmidhuber, 1997).

A partir de los años dos mil, el NLP experimentó un crecimiento explosivo debido a la disponibilidad de grandes volúmenes de datos. Herramientas como *Word2Vec*, desarrolladas por Mikolov et al. (2013), permitieron representar palabras en espacios vectoriales de alta dimensión, mejorando la representación semántica y facilitando avances en traducción automática y modelado del lenguaje, como en el caso de *Google Translate* (Koehn, 2009).

Durante la década de 2010, los avances en aprendizaje profundo, como las redes neuronales convolucionales (CNN) y modelos como el *transformer,* revolucionaron el NLP. Estos modelos, descritos por Vaswani et al. (2017), mejoraron la capacidad de manejar dependencias a largo plazo en el lenguaje y avanzaron significativamente en la comprensión y generación de texto. Además,

modelos como BERT y GPT se convirtieron en estándares para tareas de procesamiento de lenguaje natural en diversos sectores. Finalmente, en la década de 2020, modelos de lenguaje a gran escala como GPT-3, desarrollados por OpenAI, ampliaron enormemente las capacidades del NLP, permitiendo una generación de texto coherente y contextualmente precisa. El NLP continúa avanzando a un ritmo acelerado, con investigaciones y mejoras en curso que amplían los límites del entendimiento y la generación hacia capacidades más sofisticadas y matizadas.

2.3 Proceso para el análisis de sentimiento con NLP

Para abordar los métodos de análisis de sentimientos utilizando NLP y *machine learning*, se requiere comprender la evolución y aplicación de estas técnicas en el análisis de datos textuales. Estas metodologías se apoyan en algoritmos avanzados que pueden manejar grandes cantidades de texto no estructurado, identificando patrones y tendencias de manera eficiente y automatizada. A diferencia de los enfoques tradicionales que dependían de análisis manuales y encuestas de satisfacción, esta tecnología ofrece una solución escalable.

Además, el análisis de sentimiento permite a las marcas de lujo entender la aprobación de sus consumidores con sus productos. Para ello deben entender emociones específicas como gusto, desagrado, amargura o insatisfacción o sorpresa expresada por los consumidores en las reseñas. En este contexto, no se han aplicado modelos emocionales universales como los propuestos por Ekman o Plutchik. Aunque útiles para estudiar emociones fundamentales humanas, al aplicarse en este contexto presentan limitaciones ya que las reseñas pueden ser complejas y sutiles.

En la actualidad, la utilización de modelos de lenguaje como BERT y GPT, mejoran significativamente la capacidad de estos sistemas para comprender el contexto y la ironía del lenguaje. Sin embargo, la precisión y eficacia de estos modelos dependen en gran medida de la calidad y diversidad del conjunto de datos usados para su entrenamiento.

2.3.1 Selección de herramientas y plataformas

La selección de herramientas y plataformas desempeñan un papel de mucha importancia en el desarrollo de este tipo de análisis. Por ello, entre las bibliotecas más destacadas se encuentran *TensorFlow* y *PyTorch*, reconocidas por su capacidad para entrenar modelos complejos de NLP. Además, *Google Collab* es una plataforma en la nube que proporciona acceso gratuito a recursos computacionales como *GPUs* y *TPUs*, lo que facilita la experimentación con estos modelos.

Es importante mencionar que Python es el lenguaje más utilizado y versátil para el análisis de datos y en general, para trabajar con datos, debido a la amplia gama de bibliotecas que permiten realizar análisis precisos. La elección entre estas herramientas y plataformas específicas depende de varios factores, como las preferencias del equipo de desarrollo, la complejidad del proyecto, los objetivos específicos y los recursos disponibles.

2.3.2 Recopilación y preparación de datos

La recopilación y preparación de datos son etapas que cumplen un rol importante en el análisis de sentimientos. Comienza con la extracción de comentarios y valoraciones de diversas fuentes en línea, como redes sociales, foros, sitios de reseñas y blogs, donde los consumidores expresan opiniones sobre productos y servicios.

Una vez obtenidos los datos, los datos crudos se someten a un proceso de limpieza para eliminar, por ejemplo, duplicados, valores nulos, información irrelevante, corrección de ortografía, normalización del texto entre otras consideraciones con la finalidad de garantizar la calidad y relevancia del conjunto de datos. La normalización es esencial ya que proporciona consistencia al formato y la codificación del texto, estandarizando elementos como mayúsculas, minúsculas, abreviaturas, espacios en blanco, emojis y caracteres especiales. Para ello, se emplean técnicas para transformar el conjunto de datos en un formato estructurado. Esto incluye la *tokenización*, que divide el texto en unidades significativas como palabras o frases, y la lematización o *stemming*, que simplifica las palabras a sus formas base. La preparación de los datos también incluye la eliminación de los *stop words* las cuales son palabras que no tienen relevancia para el análisis. La preparación de los datos también puede involucrar la creación de representaciones vectoriales del texto mediante métodos como *Word2Vec*, *TF-IDF* o *embeddings* de palabras, conservando la semántica y el contexto de las palabras.

La calidad del preprocesamiento del conjunto de datos y, en especial del texto, influye directamente en la efectividad del modelo ya que permite que los algoritmos detecten patrones significativos fácilmente.

2.3.3 Modelado y entrenamiento de algoritmos

El modelado y entrenamiento de algoritmos constituyen una fase que implica la selección de modelos de NLP. El entrenamiento de los algoritmos comienza con la configuración de los hiperparámetros adecuados el cual ayuda con el rendimiento del modelo. Para elegir estos hiperparámetros se utilizan técnicas como la validación cruzada, que evalúa cómo el modelo se comporta en diferentes divisiones del conjunto de datos y, con ello, ayuda a prevenir el sobreajuste, promoviendo así una mejor capacidad de generalización del modelo a nuevos datos.

La evaluación de los modelos se realiza utilizando métricas como precisión, *recall*, F1-Score y AUC-ROC, los cuales proporcionan una evaluación integral del rendimiento del modelo. Esta fase facilita, mediante el resultado de las métricas, selecciona el mejor modelo para el análisis de sentimiento en este contexto.

2.4 Métricas de evaluación de modelos de machine learning

Para evaluar la efectividad de los modelos de manera cuantitativa se utilizan métricas específicas las cuales incluyen, como se mencionó, la precisión, *recall*, F1-Score, AUC-ROC y, también, la matriz de confusión. Cada una de ellas otorga una evaluación al modelo, por ejemplo: la precisón mide la proporción de predicciones correctas entre todas las predicciones positivas, el *recall* la cual evalúa la proporción de verdaderos positivos correctamente identificados por el modelo, y el F1-Score que proporciona un equilibrio entre las dos anteriores. Además, el AUC-ROC evalúa la capacidad discriminativa del modelo, especialmente en la clasificación de reseñas positivas y negativas. Por último, la matriz de confusión que de manera visual permite ver qué rendimiento tiene el modelo en el conjunto de datos analizado.

2.4.1 Precisión

La precisión es la proporción de verdaderos positivos (instancias correctamente clasificadas como positivas) sobre el total de instancias clasificadas como positivas (verdaderos positivos más falsos positivos).

$$Precisión = \frac{TP}{TP\ + FP},$$

Donde:

TP (*True Positives*): Verdaderos positivos. Son los casos en los que el modelo ha predicho correctamente la clase positiva (es decir, el modelo identificó correctamente las instancias que pertenecen a la clase positiva).

FP (*False Positives*): Falsos positivos. Son los casos en los que el modelo ha predicho incorrectamente la clase positiva (es decir, el modelo identificó erróneamente como positivos los casos que en realidad son negativos)

2.4.2 Recall (sensibilidad)

El *recall* es la proporción de verdaderos positivos sobre el total de instancias que realmente son positivas (verdaderos positivos más falsos negativos).

$$Recall = \frac{TP}{TP\ + FN},$$

Donde:

TP ("True Positives"): Verdaderos positivos. Son los casos en los que el modelo ha predicho correctamente la clase positiva (es decir, el modelo identificó correctamente las instancias que pertenecen a la clase positiva).

FP ("False Positives"): Falsos positivos. Son los casos en los que el modelo ha predicho incorrectamente la clase positiva (es decir, el modelo identificó erróneamente como positivos los casos que en realidad son negativos)

2.4.3 F1-score

El F1-score es la media armónica de la precisión y el *recall*. Se calcula como

$$F1 = 2x\frac{Precisión\ x\ Recall}{Precisión + Recall},$$

Donde:

Precisión: Como se explicó anteriormente, mide la proporción de predicciones positivas correctas (TP) sobre todas las predicciones positivas (TP + FP)

Recall: Es la proporción de casos positivos que fueron correctamente identificados por el modelo

2.4.4 AUC-ROC (Área bajo la curva – Curva de operación del receptor)

AUC-ROC mide la capacidad del modelo para distinguir entre clases. La curva ROC es un gráfico de la tasa de verdaderos positivos (TPR) frente a la tasa de falsos positivos (FPR) a varios umbrales de clasificación. El AUC representa el área bajo esta curva.

$$AUC - ROC = \int_0^1 TPR(FPR)d(FPR),$$

Donde:

TPR (*True Positive Rate*), también conocido como *recall* o sensibilidad

FPR (*False Positve Rate*)

2.4.5 Matriz de confusión

Una matriz de confusión es una de las herramientas más utilizadas en el ámbito de *machine learning* ya que se utiliza para evaluar el rendimiento de modelos de clasificación. Esta matriz se presenta en formato tabular y compara las predicciones del modelo con las etiquetas reales del conjunto de datos, proporcionando una visualización clara de las clasificaciones verdaderamente positivas (TP), verdaderamente negativas (TN), falsamente positivas (FP) y falsamente negativas (FN). En el contexto de NLP, esta matriz de confusión puede mostrar cómo el modelo maneja correcta o incorrectamente las predicciones de sentimientos positivos y negativos. Esto permite la identificación de patrones de error específicos, como la tendencia del modelo a clasificar erróneamente sentimientos positivos como negativos (falsos negativos) o viceversa (falsos positivos) esto permite ajustar y optimizar el modelo para mejorar su capacidad de generalización.

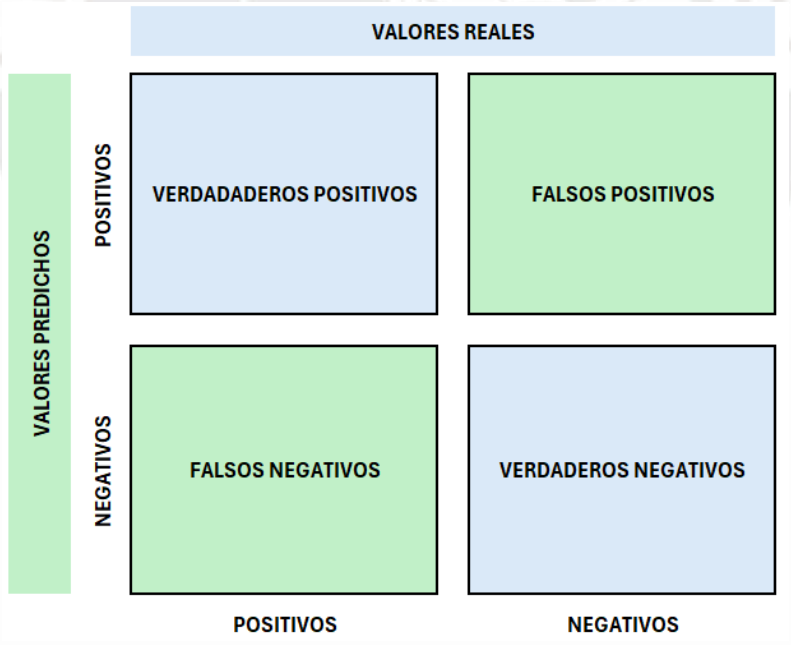


Figura 1

Matriz de confusión

2.4.6 Resumen de métricas

La evaluación de modelos en NLP utiliza diversas métricas para comprender su rendimiento. Entre estas métricas se encuentran la precisión, el *recall*, el F1-Score, el AUC-ROC y la matriz de confusión. Por ejemplo, la precisión asegura que las reseñas clasificadas como positivas realmente lo sean, evitando errores al identificar incorrectamente opiniones neutrales. Por otro lado, el *recall* captura la mayoría de las reseñas positivas presentes en el conjunto de datos, lo que permite tener una visión completa de la satisfacción del cliente. El F1-Score combina tanto la precisión como el *recall* para lograr un equilibrio entre ambos, minimizando tanto los falsos positivos como los falsos negativos en la clasificación de sentimientos. El AUC-ROC, por su parte, evalúa la capacidad del modelo para distinguir correctamente entre reseñas positivas y negativas sin depender de umbrales específicos, lo cual indica su eficiencia para esta tarea. Por último, la matriz de confusión analiza como el modelo clasifica las reseñas en términos de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos, identificando de esta manera errores específicos para mejorar la precisión de la clasificación.

2.4.7 Interpretación de resultados

En la interpretación de resultados en el análisis de sentimiento de reseñas en la industria del lujo, se utilizan apoyos como gráficos de barras, nubes de palabras para visualizar patrones claros en la opinión de los consumidores, pero también se debe tener claro que para realizar estas interpretaciones se debe tener un conocimiento profundo tanto del sector como de la compañía y de sus productos ya que esto ayuda a interpretar mejor los resultados más allá de las métricas. Las métricas, las cuales se describieron en el punto 2.4.6 ayudan a interpretar cómo un modelo se comporta frente a este tipo de datos por lo que mediante una gráfica se facilita la visualización del rendimiento relativo de estos modelos en este contexto específico. Resaltar que, las conclusiones o interpretación se basan exclusivamente en este conjunto de datos ya que, para otras aplicaciones o incluso para otros sectores esta interpretación podría no ser la adecuada.

CAPÍTULO 3 – ANÁLISIS EXPLORATORIO Y PREPROCESADO DEL CONJUNTO DE DATOS DE LUXURY BEAUTY

El análisis exploratorio y preprocesado de datos son etapas de vital importancia ya que es la base para desarrollar un modelo de análisis de sentimiento basado en procesamiento de lenguaje natural (NLP), especialmente en el contexto del conjunto de datos de *Luxury Beauty*, que se centra en productos y servicios de alta gama en la industria de belleza. Estas fases iniciales ayudan a comprender el conjunto de datos, explorando las reseñas y valoraciones que han hecho los consumidores con la finalidad de captar sus opiniones y percepciones. Para un sector, como es el del lujo, es importante que el consumidor se encuentre satisfecho con la compra realizada y, por ende, con el producto es por ello que es importante ser riguroso con el análisis y el tratamiento de los datos ya que en base a ellos se toman decisiones y, en muchos casos, se realizan inversiones con una suma importante de dinero por lo que hay que se debe tener una especial atención al detalle para poder realizar este tipo de análisis ya que impacta directamente al negocio una mala interpretación de los datos conllevaría a tener una mala reputación lo que es lo que se desea evitar.

3.1 Descripción del conjunto de datos de Luxury Beauty

El conjunto de datos que se está usando en esta tesis consiste en valoraciones y reseñas de consumidores sobre una variedad de productos de alta gama en el sector de belleza. Además, incluye múltiples registros con detalles como puntuaciones globales, número de votos, verificación de compra, fecha de valoración, identificadores de usuarios y productos, texto y resumen de valoraciones, entre otros atributos relevantes. Al realizar un análisis inicial se ve tendencias significativas: la mayoría de las valoraciones son muy positivas, con cuatro o cinco estrellas, lo que indica una alta satisfacción general con los productos adquiridos. Es importante considerar el posible efecto de valoraciones extremas en la percepción general. Además, se observa que algunos productos reciben un número desproporcionado de valoraciones, lo que refleja su popularidad y el impacto de estrategias de marketing. El análisis incluye técnicas estadísticas y visualizaciones de histogramas para examinar la distribución de calificaciones, análisis de frecuencia de palabras para identificar temas recurrentes, y un análisis preliminar de sentimiento para comprender las emociones predominantes en las reseñas. Estos hallazgos proporcionan una comprensión profunda del nivel de satisfacción de los consumidores y permite establecer una base sólida para el desarrollo de estrategias comerciales efectivas adaptadas al mercado del lujo.

3.2 Importación de librerías

```python
import os
import json
import gzip
import pandas as pd
from urllib.request import urlopen
```

Figura 2

Importación de librerías

Para la importación de librerías necesarias en el procesamiento de datos, la Figura 2 muestra uso de herramientas que permiten extraer los datos. Estos incluyen "os" para la interacción con el sistema operativo, *json* y *gzip* para la gestión de archivos *JSON* y comprimidos respectivamente, así como *pandas* para la manipulación y análisis de datos estructurados. También se utiliza *urllib.request* para abrir y descargar datos directamente desde *URLs*.

3.3 Selección de datos y filtrado de columnas para procesamiento NLP

```python
# Cargando el conjunto de datos
data = []
with gzip.open("Datasets\Luxury_Beauty.json.gz") as f:
    for line in f:
        data.append(json.loads(line.strip()))

# Convirtiendo a DataFrame
df = pd.DataFrame.from_dict(data)

# Mostrando las primeras filas del conjunto de datos
df.head()
```

Figura 3

Carga y lectura de los datos

Para extraer y organizar los datos del archivo *Luxury Beauty* desde un archivo *JSON* comprimido, la Figura 3 muestra el procedimiento para realizarlo. Inicialmente, se leen y descomprimen las líneas del archivo para convertirlas en diccionarios Python, los cuales se almacenan en una lista llamada data. Posteriormente, se convierte esta información en un conjunto de datos y, para ver que la extracción fue correcta, se muestran los primeros registros.

	overall	vote	verified	reviewTime	reviewerID	asin	reviewerName	reviewText	summary	unixReviewTime	style	image
0	2.0	3	True	06 15, 2010	A1Q6MUU0B2ZDQG	B00004U9V2	D. Poston	I bought two of these 8.5 fl oz hand cream, an...	dispensers don't work	1276560000	NaN	NaN
1	5.0	14	True	01 7, 2010	A3HO2SQDCZIE9S	B00004U9V2	chandra	Believe me, over the years I have tried many, ...	Best hand cream ever.	1262822400	NaN	NaN
2	5.0	NaN	True	04 18, 2018	A2EM03F99X3RJZ	B00004U9V2	Maureen G	Great hand lotion	Five Stars	1524009600	{'Size:': ' 3.5 oz.'}	NaN
3	5.0	NaN	True	04 18, 2018	A3Z74TDRGD0HU	B00004U9V2	Terry K	This is the best for the severely dry skin on ...	Five Stars	1524009600	{'Size:': ' 3.5 oz.'}	NaN
4	5.0	NaN	True	04 17, 2018	A2UXFNW9RTL4VM	B00004U9V2	Patricia Wood	The best non- oily hand cream ever. It heals o...	I always have a backup ready.	1523923200	{'Size:': ' 3.5 oz.'}	NaN

Figura 4

Dataframe resultante

En la Figura 4 se destaca la importancia de analizar las valoraciones y reseñas. En este contexto, se hace necesario realizar un filtrado de los datos, centrándose en las columnas *overall* y *reviewText* las cuales son las columnas que van a aportar información al análisis de sentimiento ya que contienen texto que permiten identificar la valoración de un determinado producto. Es por ese motivo que el resto de las columnas, que también tienen información valiosa, se eliminan, pero para otros análisis en contexto podrían aportar valor.

```python
#Se realiza el filtro de las columnas 'Overall' y 'reviewText'
#con la finalidad de realizar un analisis de sentimiento
sentiment_df = df[['overall', 'reviewText']].reset_index(drop=True)

sentiment_df = sentiment_df.dropna()
```

Figura 5

Filtrado de las columnas 'overall' y 'reviewtext'

En la Figura 5 se detalla el proceso de selección de las columnas *overall* y *reviewText*, las cuales como se mencionó en el punto anterior, son de intereses para este análisis. Posteriormente, se lleva a cabo la eliminación del resto de información debido a que no aportar valor en este contexto. Luego de hacer una primera depuración, el *dataframe* queda conformado por 574228 filas y 2 columna, lo que subraya una cantidad importante de reseñas y que va a permitir que los modelos que se implementan más adelante tengan mayor variedad en las entradas al modelo.

3.4 Análisis estadísticos básicos y visión general de los resultados

```python
# Estadísticas básicas de la columna "overall"
overall_stats = sentiment_df['overall'].describe()
overall_stats
```

```
count    574228.000000
mean          4.225369
std           1.297647
min           1.000000
25%           4.000000
50%           5.000000
75%           5.000000
max           5.000000
Name: overall, dtype: float64
```

Figura 6

Estadísticas básicas de la columna 'overall'

En la Figura 6 se presenta la estadística descriptiva de la columna *overall*, donde se observa que hay un total de 574228 valoraciones con una media de 4.23, lo que indica que hay cierta inclinación hacia las reseñas positivas. La desviación estándar de 1.30 sugiere cierta variabilidad en las calificaciones. Los valores oscilan entre uno y cinco, así mismo los percentiles se encuentra predominantemente entre 4 y 5, lo que está confirmando que la mayoría de las opiniones son positivas.

```python
import matplotlib.pyplot as plt

# Visualización de la distribución de las calificaciones
plt.figure(figsize=(10, 6))
sentiment_df['overall'].value_counts(sort=False).plot(kind='bar', color='skyblue')
plt.title('Distribución de Calificaciones')
plt.xlabel('Calificación')
plt.ylabel('Número de Reseñas')
plt.xticks(rotation=0)
plt.show()
```

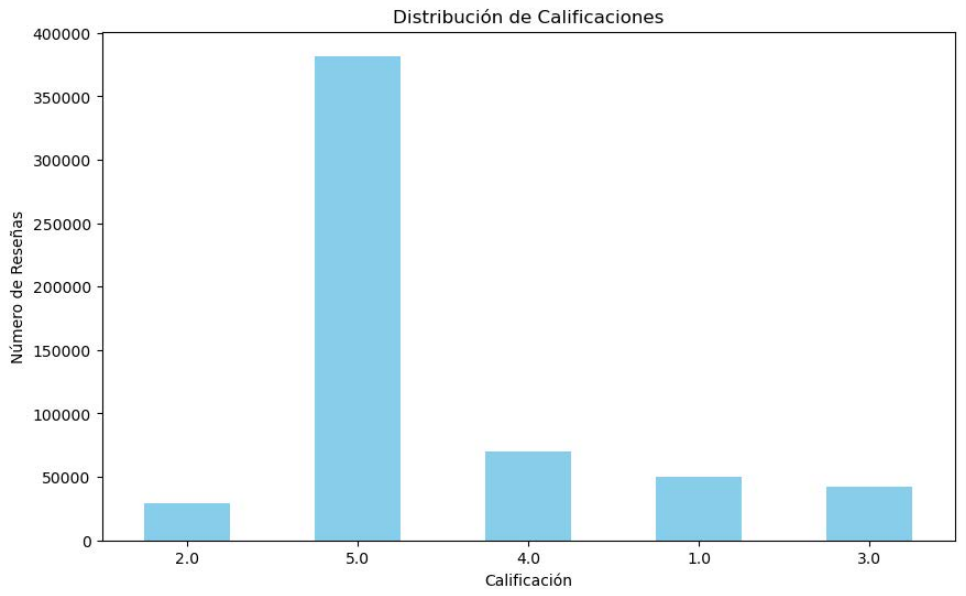
Figura 7

Fragmento de código para la visualización de la distribución de las calificaciones

En la Figura 7 se presenta un fragmento de código que genera una visualización de la distribución de calificaciones utilizando *Matplotlib*, mostrando la frecuencia de cada calificación en un gráfico de barras. Cada una de éstas representa la frecuencia de reseñas para cada puntuación.

Gráfica 1

Distribución de Calificaciones de las reseñas



En la gráfica 1 se observa que una gran parte de las reseñas reciben la calificación máxima de cinco, lo cual confirma la tendencia positiva previamente mencionada. Esta distribución resalta la importancia de realizar un análisis detallado de las reseñas para entender mejor las preferencias de los consumidores e identificar oportunidades de mejora para productos.

```python
# Calculando La longitud de cada reseña
sentiment_df['review_length'] = sentiment_df['reviewText'].apply(lambda x: len(str(x)))

# Visualizando la distribución de la longitud de las reseñas
plt.figure(figsize=(12, 6))
sentiment_df['review_length'].hist(bins=100, color='skyblue', range=[0, 2000])
plt.title('Distribución de la Longitud de las Reseñas')
plt.xlabel('Longitud de la Reseña')
plt.ylabel('Número de Reseñas')
plt.show()
```

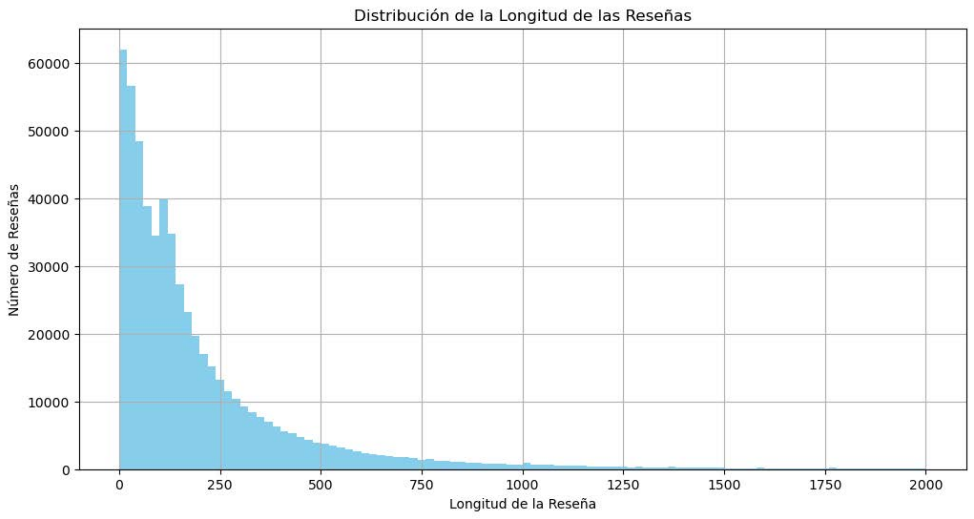
Figura 8

Fragmento de código para calcular la longitud de las reseñas de los consumidores y visualizarlas

En la Figura 8 se muestra el código empleado para calcular la longitud de cada reseña en el *dataframe sentiment_df* y configura la visualización de esta distribución mediante un histograma. Analizar la longitud de cada reseña permite interpretar que reseñas más largas tienden a contener más detalles mientras que las más cortas suelen ser más directas.

Gráfica 2

Distribución de la longitud de las reseñas de los consumidores



El Gráfico 2 muestra la distribución de la longitud de las reseñas. La mayoría, como se aprecia, son breves es decir son reseñas directas sin mayor detalle. Este patrón sugiere que los consumidores expresan sus opiniones de manera concisa, lo cual es relevante para ajustar algoritmos que se centran en textos cortos.

3.5 Proceso de balanceado - Undersampling

El *undersampling* como concepto es una técnica que se emplea para balancear conjuntos de datos cuando hay una disparidad significativa entre las clases. En este contexto y en este conjunto de datos se puede ver que mayormente las reseñas son positivas. Debido a ello, con la ayuda de esta estrategia se busca reducir el número de ejemplos de la clase mayoritaria (como las reseñas positivas) para equipararla con la clase minoritaria. Pero se debe considerar que, aunque puede mejorar el rendimiento y la capacidad de generalización del modelo, existe el riesgo de perder información crucial por lo que es importante a su vez que la interpretación de los resultados vaya de la mano con el conocimiento del negocio.

```python
# Contando el número de ejemplos para cada clase (calificación)
class_counts = sentiment_df['overall'].value_counts()

# Determinando el número de ejemplos de la clase menos representada
min_class_count = class_counts.min()

class_counts, min_class_count
```

Figura 9

Fragmento de código para calcular el número de ejemplos para cada clase

En la Figura 9, se determina cuántos ejemplos hay en cada categoría de calificación, identificando cuál tiene menos representaciones. Esto resulta importante para aplicar técnicas de balanceo como el que se mencionó en el punto anterior, el *undersampling*, pues ayuda a entender la distribución de clases y evitar que el modelo se concentre demasiado en la categoría mayoritaria. Conocer el número mínimo de ejemplos permite establecer un límite en la cantidad de datos que se usan en las clases más comunes, logrando así un conjunto de datos más equilibrado.

```
(overall
 5.0    381823
 4.0     70451
 1.0     50477
 3.0     41975
 2.0     29502
 Name: count, dtype: int64,
 29502)
```

Figura 10

Cantidad de ejemplos para cada calificación de la columna 'overall'

En la Figura 10, se observa un desequilibrio en las calificaciones, dominadas en gran medida por evaluaciones de cinco estrellas, lo que sugiere una percepción positiva de los productos. Sin embargo, para analizar los sentimientos de manera equilibrada, es necesario prestar atención a la clase con menos ejemplos, que en ese caso son las reseñas de dos estrellas, con 29502 registros. Si no se corrige este desequilibrio es posible que el modelo de análisis de sentimientos se incline hacia las opiniones positivas. Es por ello por lo que se emplea la técnica del *undersampling*.

```python
# Realizando el undersampling
balanced_data = []

# Para cada calificación única, seleccionamos aleatoriamente 'min_class_count' ejemplos
for rating in class_counts.index:
    sample = sentiment_df[sentiment_df['overall'] == rating].sample(n=min_class_count, random_state=42)
    balanced_data.append(sample)

# Combinando los datos balanceados
balanced_df = pd.concat(balanced_data, axis=0)

# Verificando la distribución de clases en el conjunto de datos balanceado
balanced_counts = balanced_df['overall'].value_counts()
balanced_counts
```

Figura 11

Fragmento de código para aplicar el método de 'undersampling'

En la Figura 11, se implementa el proceso de *undersampling* para igualar la cantidad de reseñas en cada calificación, tomando una muestra aleatoria del mismo tamaño que la clase menos representada. Se emplea un estado aleatorio fijo para garantizar que el proceso pueda repetirse de forma confiable. Al final de este proceso, todas las muestras se combinan para formar un nuevo conjunto de datos con una distribución uniforme de calificaciones. Esto permite evitar que cualquier modelo de aprendizaje automático se sesgue hacia las clases más frecuentes y favorece que aprenda de manera más equilibrada.

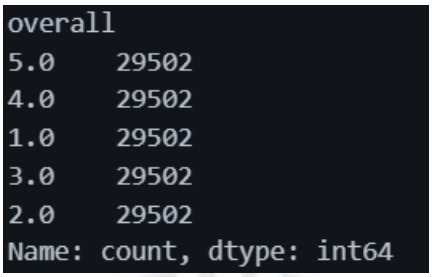


Figura 12

Cantidad de ejemplos para cada calificación de la columna 'overall' luego del 'undersampling'

En la Figura 12 se observa que, tras aplicar el *undersampling*, cada calificación (de 1 a 5 estrellas) queda representada con 29502 registros, obteniendo así un conjunto de datos equilibrado. Esta igualdad en la distribución es relevante para evitar sesgos en el análisis de sentimientos, ya que permite que los modelos de *machine learning* aprendan de todas las calificaciones con la misma relevancia, sin inclinarse hacia la clase que antes predominaba.

3.6 Etiquetado de comentarios positivos y negativos

Para mejorar el análisis de sentimiento se debe etiquetar las reseñas como positivas o negativas- En el contexto de productos de lujo, las calificaciones de tres, cuatro y cinco estrellas se agrupan como positivas, considerando que incluso una calificación de tres aún refleja cierto nivel de satisfacción en un mercado tan exigente. En contraste, las calificaciones de una y dos estrellas se clasifican como negativas, indicando experiencias claramente insatisfactorias. Este enfoque simplifica la tarea del modelo al diferenciar entre reseñas que muestran satisfacción, aunque sea mínima, y aquellas que destacan problemas significativos. Es importante recalcar que el sector de lujo básicamente se centra en analizar el nivel de satisfacción por lo que esta consideración corresponde a necesidades del negocio.

```
# Creando un nuevo DataFrame a partir de balanced_df
labeled_df = balanced_df.copy()

# Etiquetando los datos en el nuevo DataFrame
# 0: Es una reseña negativa
# 1: Es una reseña positiva
labeled_df['label'] = labeled_df['overall'].apply(lambda x: 0 if x in [1, 2] else 1)

# Mostrando las primeras filas del nuevo conjunto de datos etiquetado
labeled_df.head()
```

Figura 13

Fragmento de código para etiquetar los comentarios

En la Figura 13 se crea un nuevo *dataframe* llamado *labeled_df*, donde las reseñas con calificaciones de uno o dos reciben la etiqueta de cero (negativas), y las de 3 a 5 obtienen la etiqueta uno (positivas).

```python
# Filtrando el DataFrame para quedarse solo con las columnas 'reviewText' y 'label'
final_df = labeled_df[['reviewText', 'label']]

# Mostrando las primeras filas del DataFrame final
final_df.head()
```

Figura 14

Fragmento de código seleccionar las columnas 'reviewText' y 'label'

En la Figura 14 se seleccionan únicamente las columnas *reviewText* y *label* para dejar el conjunto de datos preparado para emplearlos con los modelos de *machine learning*. Después, se revisa la estructura final verificando que cada reseña tenga asignada su etiqueta de sentimiento. Este paso permite que los modelos se centren solo en la información necesaria al entrenar. En la Figura 15 se muestra el resultado de este proceso.

	reviewText	label
530069	Great product . Life saver for people who stru...	1
441589	World great !	1
63941	ultra compact and works very well	1
364878	It's is my long-time favorite and I'm not trad...	1
559515	This is a very light but masculine, slightly c...	1

Figura 15

Muestra del dataframe resultante

3.7 Cardinalidad

La cardinalidad, como concepto, hace referencia al número de elementos únicos es un conjunto de datos específico. Por ejemplo, en un corpus de texto, la cardinalidad representa la cantidad de palabras distintas presentes por lo que una alta cardinalidad implica una amplia variedad de términos, lo cual puede complicar el proceso de modelado aumentando las características a analizar. En este contexto, como la clasificación de reseñas, cada palabra única se convierte en una característica, incrementando la dimensionalidad del problema. Para abordar esto, se emplea la lematización, que reduce las variantes de una palabra a su forma básica (por ejemplo, convirtiendo "corriendo" en "correr"), simplificando así el modelo.

```python
import nltk
from nltk.tokenize import word_tokenize

nltk.download('punkt')

# Tokenizando las reseñas y calculando el conjunto de palabras únicas
all_reviews = " ".join(final_df['reviewText'].astype(str))
tokens = word_tokenize(all_reviews)
unique_tokens = set(tokens)

# Calculando la cardinalidad
cardinality = len(unique_tokens)
cardinality
```

Figura 16

Fragmento de código para la tokenización de reseñas y el cálculo de palabras únicas

En la Figura 16 se muestra cómo se agrupan todas las reseñas en un único texto y, mediante *word_tokenize* de *NLTK*, se separan en palabras individuales. Luego, al convertir esta lista en un conjunto, se eliminan duplicados, resultando 73082 palabras únicas, valor que indica la cardinalidad o variedad lingüística de las reseñas como se muestra en la Figura 17.

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\jeffe\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!

73082
```

Figura 17

Resultado de la cardinalidad

Esta alta cardinalidad muestra una amplia diversidad en el vocabulario, lo cual puede complicar el proceso de entrenamiento de modelos debido a la extensión del espacio de características. Para abordar esto, por lo general, se utilizan técnicas de reducción de dimensionalidad como el análisis de componentes principales (PCA) o métodos más avanzados de NLP que incorporan mecanismos de atención para destacar los términos más relevantes. Esta reducción se realiza en la etapa de preentrenamiento asegurando que los modelos tengan la información más concisa lo cual mejora su nivel de interpretación de los resultados y aporta a que estos puedan clasificar las reseñas como positivas o negativas según corresponda. En la presente tesis no se emplearon estos métodos debido a que la dimensionalidad o, columnas disponibles, es la estándar.

```python
from nltk.probability import FreqDist

# Calculando la frecuencia de las palabras
freq_dist = FreqDist(tokens)

# Obteniendo las 20 palabras más comunes
most_common_words = freq_dist.most_common(20)

# Preparando los datos para la visualización
words, frequencies = zip(*most_common_words)

# Visualizando las palabras más comunes
plt.figure(figsize=(15, 8))
plt.bar(words, frequencies, color='skyblue')
plt.title('20 Palabras Más Frecuentes')
plt.xlabel('Palabras')
plt.ylabel('Frecuencia')
plt.xticks(rotation=45)
plt.show()
```

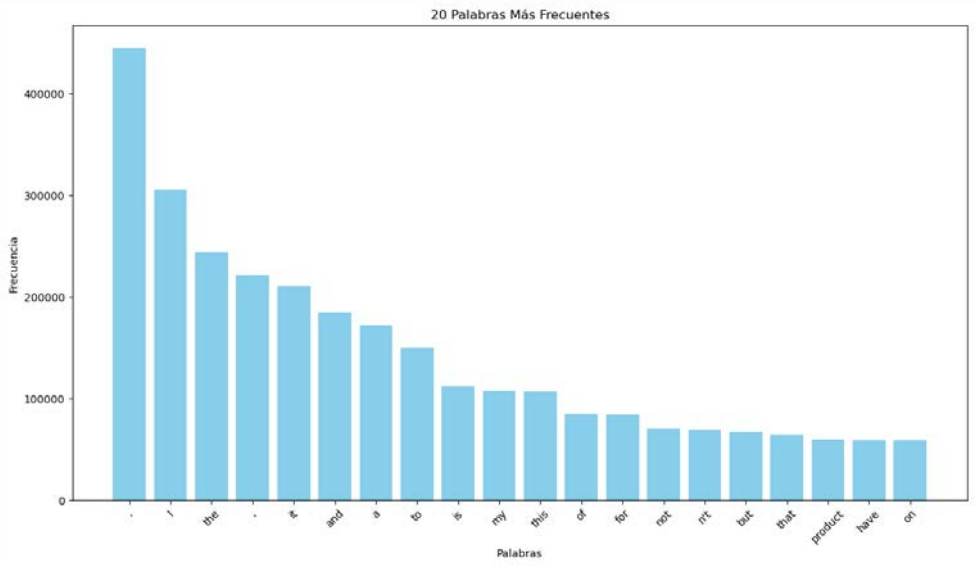
Figura 18

Fragmento de código para calcular las frecuencias de las palabras

En la Figura 18 se calcula la frecuencia de todas las palabras en *tokens* usando *FreqDist* de *NLTK* y luego se seleccionan las veinte más comunes para ser visualizadas en un gráfico de barras (palabras en el X y frecuencias en el eje Y). Este enfoque ofrece un panorama claro de los términos más utilizados en las reseñas.

Gráfica 3

Distribución de las 20 palabras más frecuentes en las reseñas



La Gráfica 3 muestra las veinte palabras más usadas en las reseñas. Es frecuente encontrar palabras de conexión como "*the*", "*and*" o "*a*", típicas en texto sin limpiar, que suelen eliminarse en el preprocesamiento por su escasa utilidad analítica. Este paso es implementado en el preprocesamiento de los datos.

3.8 Wordcloud de comentarios positivos y negativos

El proceso de crear una nube de palabras para reseñas positivas y negativas proporciona una representación visual inicial que destaca los términos más frecuentes en cada grupo. Esta fase inicial tiene la finalidad de obtener una vista panorámica de los términos más recurrentes, sin profundizar aún en un análisis detallado del contenido de las reseñas. Sin embargo, este proceso sienta las bases para un análisis más detallado centrado en adjetivos y expresiones que pueden mejorar la precisión del modelo. En la Figura 19 se puede ver cómo se implementa esta nube de palabras (*word cloud*).

```python
from wordcloud import WordCloud

# Filtrando las reseñas negativas y positivas
negative_reviews = " ".join(final_df[final_df['label'] == 0]['reviewText'].astype(str))
positive_reviews = " ".join(final_df[final_df['label'] == 1]['reviewText'].astype(str))

# Creando la nube de palabras para reseñas negativas
negative_wordcloud = WordCloud(background_color="white", width=800, height=400).generate(negative_reviews)

# Visualizando la nube de palabras de reseñas negativas
plt.figure(figsize=(12, 6))
plt.imshow(negative_wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title('Nube de Palabras para Reseñas Negativas')
plt.show()
```

Figura 19

Fragmento de código para elaborar la nube de palabras o 'word cloud' para reseñas negativas

Así mismo, lo que realiza esta operación es concentrar todas las reseñas negativas en un solo texto para que se pueda visualizar de manera clara las palabras o términos que destacan en este conjunto de datos.

Gráfica 4

Gráfica de nube de palabras o 'wordcloud' para reseñas negativas

La Gráfica 4 muestra la nube de palabras para reseñas negativas, donde términos como "use", "*bottle*", "*money*" y "*color*" son prominentes, indicando posibles aspectos que causan insatisfacción. Además, palabras como "*disappointed*" o "*return*" reflejan experiencias negativas claras. Como se comentó antes este tipo de representación facilita identificar rápidamente áreas específicas que necesitan atención y mejoras.

```
# Creando la nube de palabras para reseñas positivas
positive_wordcloud = WordCloud(background_color="white", width=800, height=400).generate(positive_reviews)

# Visualizando la nube de palabras de reseñas positivas
plt.figure(figsize=(12, 6))
plt.imshow(positive_wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title('Nube de Palabras para Reseñas Positivas')
plt.show()
```

Figura 20

Fragmento de código para elaborar la nube de palabras o 'word cloud' para reseñas positivas

En la Figura 20 se genera una nube de palabras agrupando todas las reseñas positivas en un solo texto lo que ayuda a tener una visión rápida de los términos más comunes en las experiencias satisfactorias de los clientes

Gráfica 5

Gráfica de nube de palabras o 'wordcloud' para reseñas positivas

La Gráfica 5 exhibe una nube de palabras correspondientes a reseñas positivas, donde se destacan términos como "*love*", "*great*", "*best*" y "*amazing*", lo que indica una percepción positiva de los productos. Así mismo, palabras relacionadas con la efectividad, como "*work*", y características como "*scent*" y "*smooth*", son frecuentes, mostrando las cualidades más valoradas en estos productos.

Como se puede notar estas palabras en otro contexto pueden tener otro significado es porque requiere tener un conocimiento del negocio, del sector, de los productos para poder realizar un análisis que tenga sentido y coherencia.

3.9 N-Grams

Según Bird, Loper y Klein (2009) así como Jurafky y Martin (2009), un N-grama se define como una secuencia de N palabras que aparecen contiguas en un texto. Por ejemplo, un bigrama consiste en dos palabras consecutivas y un trigrama en tres. El análisis de N-gramas ayuda a descubrir patrones que podrían no ser evidentes al examinar términos individuales, ya que ciertas combinaciones de palabras adquieren significados específicos al aparecer juntas (Koehn, 2010). Este procedimiento permite capturar el contexto de las frases, algo que el análisis de palabras individuales no logra por sí solo. Esto resulta útil porque, al considerar varios términos consecutivos, se capta mejor el contexto de cada frase, algo que las palabras aisladas no logran. Así, los modelos de análisis de sentimientos pueden diferenciar mejor las opiniones y los matices en los textos. Por ejemplo, en la frase "Jefferson come helado todos los días", los bigramas serían: "Jefferson come", "come helado", "helados todos", "todos los" y "los días". Para trigramas, se tendría: "Jefferson come helado", "come helado todos", "helado todos los" y "todos los días". Al identificar estos grupos de palabras consecutivas, se logra entender mejor el contenido y procesar con mayor detalle.

```python
from nltk.util import bigrams, trigrams

# Generando bigramas
bigrams_list = list(bigrams(tokens))

# Calculando la frecuencia de los bigramas
bigram_freq = FreqDist(bigrams_list)

# Obteniendo los 20 bigramas más comunes
most_common_bigrams = bigram_freq.most_common(20)

# Preparando los datos para la visualización
bigram_words, bigram_frequencies = zip(*most_common_bigrams)
bigram_words = [" ".join(bigram) for bigram in bigram_words]

# Visualizando los bigramas más comunes
plt.figure(figsize=(15, 8))
plt.bar(bigram_words, bigram_frequencies, color='skyblue')
plt.title('20 Bigramas Más Frecuentes')
plt.xlabel('Bigramas')
plt.ylabel('Frecuencia')
plt.xticks(rotation=45)
plt.show()
```

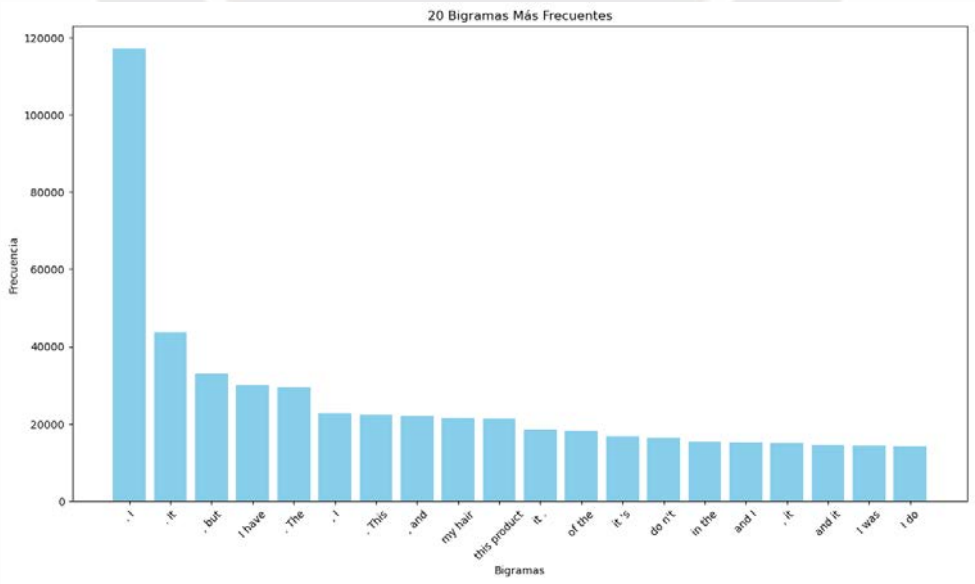
Figura 21

Fragmento de código para generar la gráfica de la frecuencia de los bigramas

En la Figura 21 se muestra el procedimiento para identificar y visualizar los bigramas más frecuentes, utilizando la biblioteca *NLTK* para generar todas las combinaciones de dos palabras consecutivas. Luego, se calcula la frecuencia de cada uno para determinar los pares más comunes. La Gráfica 6 presenta los veinte bigramas con mayor frecuencia por lo que nos ayuda a identificar patrones lingüísticos y comprender mejor los temas principales en las reseñas.

Gráfica 6

Frecuencia de los veinte bigramas más frecuentes



```python
# Generando trigramas
trigrams_list = list(trigrams(tokens))

# Calculando la frecuencia de los bigramas
trigram_freq = FreqDist(trigrams_list)

# Obteniendo los 20 bigramas más comunes
most_common_trigrams = trigram_freq.most_common(20)

# Preparando los datos para la visualización
trigram_words, trigram_frequencies = zip(*most_common_trigrams)
trigram_words = [" ".join(trigram) for trigram in trigram_words]

# Visualizando los bigramas más comunes
plt.figure(figsize=(15, 8))
plt.bar(trigram_words, trigram_frequencies, color='skyblue')
plt.title('20 Trigramas Más Frecuentes')
plt.xlabel('Trigramas')
plt.ylabel('Frecuencia')
plt.xticks(rotation=45)
plt.show()
```

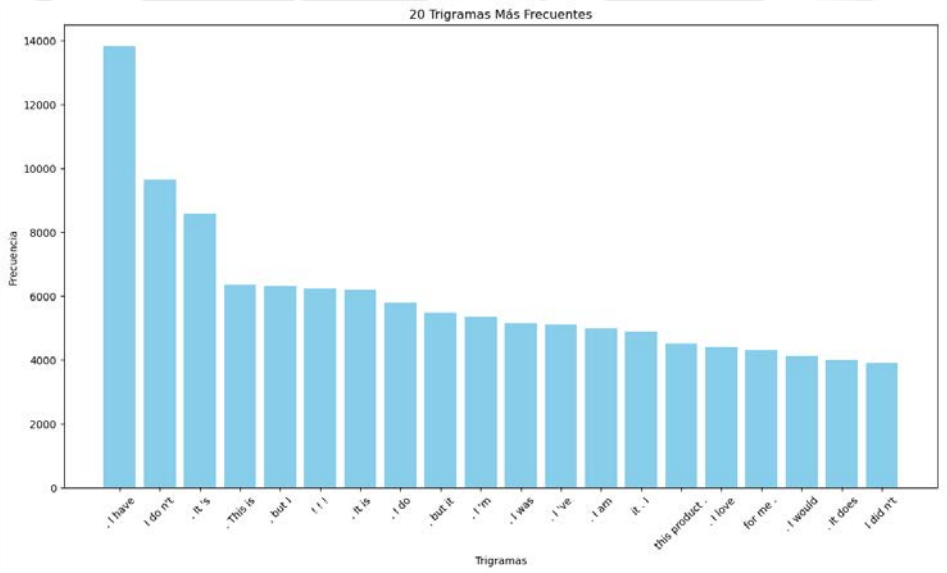
Figura 22

Fragmento de código para generar la gráfica de la frecuencia de los trigramas

En la Figura 22 se describe, de igual manera que en el caso de los bigramas, el proceso de generación y conteo de trigramas. La Gráfica 6 presenta los veinte trigramas con mayor frecuencia por lo que nos ayuda a identificar patrones lingüísticos y comprender mucho más los temas principales en las reseñas de manera resumida.

Gráfica 7

Frecuencia de los veinte trigramas más frecuentes



Se puede observar, en la Gráfica 7, que sobresalen expresiones como ". *I Have*", "*I do n't*", "*It 's*" y ". *This is*", lo que sugiere un tipo de discurso personal y subjetivo, propio de reseñas o narraciones cotidianas. Cabe señalar que en esta etapa no se ha realizado aún la limpieza de datos, por lo que algunos trigramas incluyen signos de puntuación. Dicha limpieza es un paso posterior para refinar la información antes de adentrarse en un análisis de sentimiento más profundo.

Esta alta presencia de pronombres en primera persona ("*I have*", "*I don't*", "*i'm*") y formas auxiliares ("*It's*", "*I would*", "*I did not*") apunta a un estilo narrativo introspectivo y descriptivo. Además, la mención de "*this product*" o "*for me*" indica que el texto se está centrando en la evaluación de los productos o experiencias desde una perspectiva individual. Además, hacia la

derecha del gráfico se ve una disminución progresiva en la frecuencia de aparición de los trigramas, lo que concuerda con la Ley de Zipf (Zipf, 1949), la cual plantea que la frecuencia de un término en el lenguaje natural disminuye de manera inversamente proporcional a su posición en el ranking. Aunque este principio se aplica de manera tradicional a palabras aisladas, puede extenderse a conjuntos de dos o tres palabras consecutivas (bigramas o trigramas).

3.10 Word embedding

Los *word embeddings* son una forma de codificar palabras en vectores numéricos, donde cada dimensión captura distintos aspectos semánticos y contextuales de forma más precisa que los modelos de representación tradicionales, como la "bolsa de palabras" (Mikolov, Chen, Corrado, & Dean, 2013). Esta aproximación ubica términos con significados afines en regiones cercadas de un espacio vectorial, lo que mejora la capacidad de los modelos de procesamiento de lenguaje natural (NLP) para manejar matices de similitud y contexto (Pennington, Socher, & Manning, 2014). El impacto de estos *embeddings* se ha demostrado en tareas como la traducción automática, la clasificación de textos y el análisis de sentimientos. Además, el uso de *embeddings* preentrenados ayuda a la transferencia de conocimiento y facilita el ajuste de modelos avanzados de aprendizaje profundos, incluidos los que emplean arquitecturas recurrentes y de atención.

```python
from gensim.models import Word2Vec

# Preparando el texto para el entrenamiento de Word2Vec (lista de listas de tokens)
sentences = [word_tokenize(review) for review in final_df['reviewText'].astype(str)]

# Entrenando el modelo Word2Vec
w2v_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)
w2v_model.train(sentences, total_examples=len(sentences), epochs=10)
```

Figura 23

Fragmento de código entrar el modelo 'Word2Vec'

En la Figura 23 se presenta el proceso de entrenamiento de un modelo de *Word2Vec* utilizando la biblioteca *gensim* de *Python*. Este método genera *embeddings*, representaciones vectoriales densas, a partir del contexto que se encuentra en la columna *reviewText*.

La elección de *Word2Vec* sobre *GloVe* o "*FastText*" se basa en las características específicas del conjunto de datos que se está trabajando. Se debe destacar que *Word2Vec* captura la semántica de palabras aparecen cercanas entre sí. Además, su rapidez de entrenamiento permite ciclos de iteración ágiles. Por otro lado, *GloVe* pone foco en analizar matrices de coocurrencia global, lo que, para esta aplicación, resulta ser no tan efectivo con fragmentos de texto más específicos y reducidos. *FastText*, por su parte, maneja palabras desconocidas mediante subunidades, haciendo que, para este contexto, no sea la mejor elección para el objetivo final.

```
(55641860, 79347590)
```

Figura 24

Resultado del modelo 'Word2Vec'

En la Figura 24 se muestra la salida (55641860, 79347590), que refleja, la cantidad de palabras que efectivamente han contribuido a la actualización de vectores y el total de palabras presentes en el corpus, respectivamente. Analizando este resultado se ve que las 55641860 son utilizadas para ajustar los *embeddings*, mientras que las 79347590 palabras representan el total encontrado

en el conjunto de entrenamiento. Esta discrepancia es común debido a parámetros como *min_count*, que descarta palabras con frecuencia muy baja, y la exclusión de *stop words* por ende el número de palabras realmente utilizadas en el entrenamiento es menor que el total disponible de palabras.

```python
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np

# Palabras de interés
words_of_interest = ["product", "hair", "smell", "skin", "love"]
all_words = []
all_vectors = []

# Obtener palabras similares y sus vectores
for word in words_of_interest:
    similar_words = w2v_model.wv.most_similar(word, topn=10)
    all_words.extend([word[0] for word in similar_words])
    all_vectors.extend([w2v_model.wv[word[0]] for word in similar_words])

# Convertir all_vectors a un array de NumPy
all_vectors_np = np.array(all_vectors)

# Usar t-SNE para reducción de dimensionalidad
tsne = TSNE(n_components=2, random_state=0)
word_vectors_2d = tsne.fit_transform(all_vectors_np)

# Visualización
plt.figure(figsize=(12, 8))
plt.scatter(word_vectors_2d[:, 0], word_vectors_2d[:, 1], edgecolors='k', c='blue')
for word, (x, y) in zip(all_words, word_vectors_2d):
    plt.text(x, y, word)
plt.show()
```

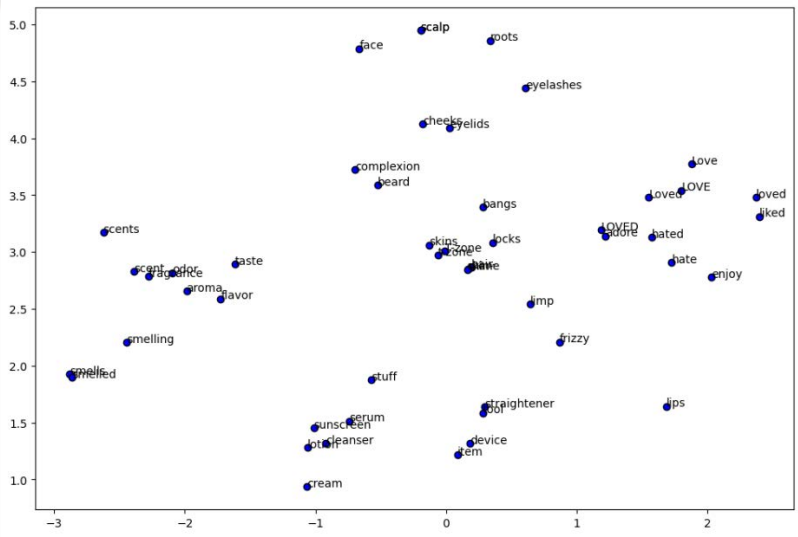
Figura 25

Fragmento de código para utilizar el algoritmo 't-sne'

En la Figura 25 se presenta cómo se aplica el algoritmo *t-SNE* de la biblioteca *Scikit-Learn* para visualizar las relaciones entre los *word embeddings* obtenidos del modelo *Word2Vec* previamente entrenado.

Gráfica 8

Visualización 2D de los 'word embeddings'



En la Gráfica 8 se observa cómo las palabras se agrupan en diferentes regiones del gráfico, reflejando de esta manera la capacidad del modelo para relacionar sus significados a partir de la reseña de los consumidores. Es importante destacar que, en esta fase inicial las palabras aparecen en su forma original sin aplicar la lematización ni el *stemming*, aspectos que se abordan en pasos

posteriores. Hay que mencionar que, la proximidad entre las palabras en el gráfico indica que sus representaciones vectoriales son similares en el espacio original. Así mismo, la técnica *t-SNE* es estocástica, lo que significa que diferentes ejecuciones pueden resultar en gráficas ligeramente distintos por lo que se establece el parámetro de *random_state* a cero para asegurar la reproducibilidad de los resultados.

3.11 Almacenamiento del conjunto de datos balanceado

Un "conjunto de datos balanceado" hace referencia a que el grupo de datos han sido ajustados para que todas las categorías estén representadas de manera equitativa. Como se mencionó esto se realizó mediante el *undersampling*.

Figura 26

Fragmento de código para almacenar el dataframe procesado

En la Figura 26 se almacena el resultado en formato csv para evitar que en pasos posteriores se vuelvan a reprocesar los datos.

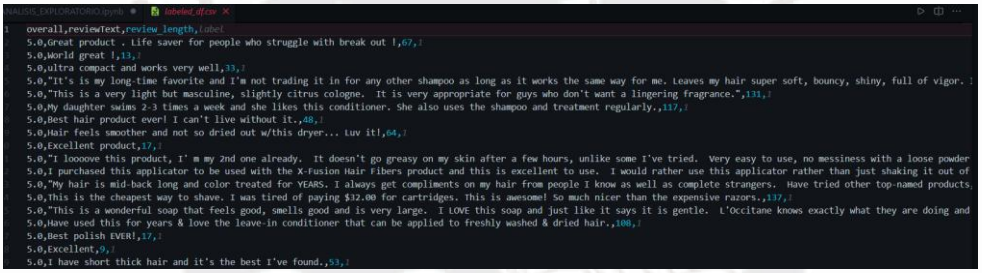


Figura 27

Muestra del dataframe procesado

En la Figura 27 se observa un ejemplar del archivo csv generado. Se observa que contiene las columnas necesarias para realizar un análisis de sentimiento como *overall*, *reviewText*, *review_length* y *label*. Además, se puede apreciar que las reseñas varían desde comentarios generales hasta descripciones detalladas de experiencias con los productos usados por los clientes. Esta estructura ayuda al análisis de sentimiento y la clasificación de textos, ya que se destacan tanto la calificación del producto como la extensión de la reseña para extraer información relevante.

CAPÍTULO 4 – PROCESADO DEL CONJUNTO DE DATOS DE LUXURY BEAUTY

El procesamiento de datos en el campo del NLP es un paso importante para lograr un análisis de sentimiento de calidad o, en otras palabras, acertar en la mayor parte de veces cuando un sentimiento es positivo o negativo. El procedimiento comienza dividiendo el texto en unidades más pequeñas, conocidas como *token*, para poder trabajar con cada palabra o símbolo por separado. Luego, se eliminan los *stop words* (palabras muy comunes como "y", "el" o "de") que no aportan información útil, lo que reduce el "ruido" en los datos. Posteriormente, se realiza la lematización, que consiste en transformar las palabras a su forma base facilitando agrupar palabras que tienen significados similares. Por último, mediante la vectorización, se convierten los tokens a números a través de técnicas como *TF-IDF* o métodos de *embeddings* (por ejemplo, *Word2Vec* y *Glove*). En conjunto, estos pasos se realizan para preparan los datos de forma adecuada ya que con ello se mejora tanto la calidad del análisis como el rendimiento de los modelos de análisis de sentimientos o *machine learning*.

4.1 Lectura y validación del conjunto de datos

```python
import pandas as pd

# Intentando leer el archivo ignorando las líneas con errores
try:
    labeled_df = pd.read_csv('labeled_df.csv', on_bad_lines='skip')
    successful_read = True
except Exception as e:
    successful_read = False
    error_message = str(e)

successful_read, labeled_df.shape if successful_read else error_message
```

Figura 28

Fragmento de código para leer el archivo en formato csv

En la Figura 28 se muestra un ejemplo de cómo se carga un archivo en formato csv, en este caso, "labeled_df.csv" la cual fue generado en el punto 3.1. El resultado se muestra en la Figura 29.

```
(True, (147510, 4))
```

Figura 29

Resultado de la lectura del archivo en formato csv

Esto significa que la lectura del archivo *labeled_df.csv* se realizó sin problemas y se cargó correctamente en un *dataframe*. El valor True, además, confirma que la lectura fue exitosa. Por otra parte, se aprecia que el conjunto de datos tiene una cantidad importante de registros lo que va a facilitar el entrenamiento del modelo de análisis de sentimiento. También podemos revisar los primeros registros de este *dataframe* utilizando el método *head ()* como se muestra en la Figura 30.

```
labeled_df.head()
```

Figura 30

Fragmento de código para mostrar los primeros registros del dataframe

Esto ofrece una inspección inicial que confirma que los datos se han cargado correctamente y ayuda a entender la estructura del conjunto antes de realizar el procesado de datos. En la Figura 31 se muestra el resultado.

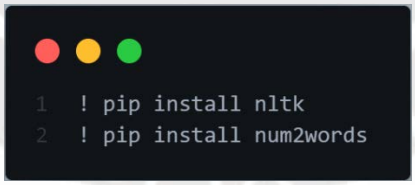
	overall	reviewText	review_length	label
0	5.0	Great product . Life saver for people who stru...	67	1
1	5.0	World great !	13	1
2	5.0	ultra compact and works very well	33	1
3	5.0	It's is my long-time favorite and I'm not trad...	442	1
4	5.0	This is a very light but masculine, slightly c...	131	1

Figura 31

Muestra del dataframe labeled_df

La columna *overall*, muestra que todas las calificaciones están en los cinco puntos, lo que sugiere reseñas positivas. En el análisis de sentimiento, estos números se usan como etiquetas para entrenar modelos de clasificación supervisada. La columna *reviewText*, contiene el texto de las reseñas, que es el elemento principal en el análisis del lenguaje. Así mismo, la columna *review_length*, indica la cantidad de caracteres de cada reseña. Esta métrica puede ser usada para investigar la extensión de una reseña que está relacionada con la intensidad del sentimiento expresado, ya que reseñas más largas pueden ofrecer información más detallada. Por último, la columna *label*, la cual contiene un valor binario entre uno o cero indica cuando una reseña es positiva o negativa.

4.2 Definición de la función de procesado



```
! pip install nltk
! pip install num2words
```

Figura 32

Instalación de librerías 'nltk' y 'num2words'

En la Figura 32 se realiza la instalación de librerías en este caso *NLTK* (*Natural Language Toolkit*) y *Num2Words*. *NLTK* ofrece funciones para trabajar con texto, como dividirlos en palabras (*tokenización*), identificar partes del discurso, extraer nombres propios y aplicar algoritmos para clasificar o analizar sentimientos. Por otro lado, *Num2Words* se encarga de convertir números en palabras, esto permite tratar los números como cualquier otra palabra.

Figura 33

Importación de la biblioteca 'nltk'

En la Figura 33 se describe brevemente módulos esenciales que se descargan con *NLTK* para trabajar con lenguaje natural. Al usar *punkt*, se obtiene un paquete para dividir el texto en palabras o frases permitiendo procesar el texto de manera que se respeten las características propias del idioma. Luego *stop words*, accede a listas de palabras muy comunes (como "y", "es", "en") que generalmente no aportan información relevante, eliminar estas palabras ayuda a centrarse en aquellas que realmente tienen significado para el análisis. Así mismo, con *wordnet* se obtiene una base de datos léxica la cual organiza las palabras por sinónimos, antónimos y relaciones de significado, este recurso es útil para entender mejor el contexto y desambiguar el significado de las palabras. Estos tres módulos, en conjunto, forman una base sólida para el preprocesamiento del texto.

Figura 34

Importación de la biblioteca unicodedata

En la Figura 34 se observa las bibliotecas más usadas para el preprocesamiento de datos. Por ejemplo, se utiliza la biblioteca *unicodedata* para asegurar que el texto se maneje de manera uniforme, convirtiendo los caracteres a una forma estándar y evitando problemas de codificación. Para dividir el texto en partes más pequeñas, se usa *word_tokenize* que separa el contenido en símbolos. Además, *RegexpTokenizer* permite hacer esta división de manera más personalizada, empleando patrones específicos para extraer las partes de interés.

La herramienta *Word2NetLemmatizer* reduce las palabras a su forma base o lema, de modo que variantes de una misma palabra se agrupen bajo un mismo significado. Por último, *FreqDisct* se usa para contar la frecuencia de las palabras en el corpus, ayudando a identificar cuáles son las más importantes y a detectar patrones en el uso del lenguaje.

```python
def preprocess_text(text):
    # Eliminamos acentos
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')

    # Tokenizamos usando una simple división basada en espacios en blanco y convertimos a minúsculas
    tokenized_text = word_tokenize(text.lower())

    # Eliminamos signos de puntuación, caracteres especiales
    tokenized_text = [word for word in tokenized_text if word.isalnum()]

    # Eliminamos stop words
    stop_words = set(stopwords.words('english'))
    tokenized_text = [word for word in tokenized_text if word not in stop_words]

   # Lemmatización
    lemmatizer = WordNetLemmatizer()
    tokenized_text = [lemmatizer.lemmatize(word).strip() for word in tokenized_text]

    # Convertimos números a palabras
    for i, word in enumerate(tokenized_text):
        if word.isdigit():
            tokenized_text[i] = num2words(int(word), lang='en')

    # Calculamos las palabras más frecuentes y las eliminamos
    word_freq = FreqDist(tokenized_text)
    most_common_words = set(word for word, freq in word_freq.most_common(5))

    # Unimos el resultado en un texto limpio
    clean_text = ' '.join(tokenized_text)

    return clean_text
```

Figura 35

Fragmento de código para procesamiento de texto

En la Figura 35 se desarrolla la función *preprocess_text* la cual se va a encargar de normalizar las reseñas. Primero, se normaliza el texto usando una función que convierte los caracteres a una forma estándar y elimina acentos, lo que permite evitar problemas de codificación. Luego, se convierte el texto a minúsculas y se divide en palabras (*tokenización*), generando de esta forma una lista de términos que se pueden analizar fácilmente. A continuación, se filtran los tokens para eliminar la puntuación y caracteres especiales, quedándose solo con los términos alfanuméricos. Además, se eliminan las palabras muy comunes (*stop words*) que, como se mencionó anteriormente, no aportan mucho significado al análisis. Posteriormente, se realiza la lematización, que va a reducir las palabras a su forma base; por ejemplo, diferentes variantes de una misma palabra se unifican en un solo término. Esto es preferible al *stemming*, ya que preserva el significado correcto de cada palabra de manera contraria al *stemming* que puede ser muy agresivo y, a menudo, produce raíces que no son palabras completas ni significativas en sí mismas por lo que puede dificultar la interpretación del texto ya que se pierde parte del significado original de cada término.

Al final de la función, se eliminan las palabras que aparecen con demasiada frecuencia, pues estos pueden distorsionar el análisis, y se vuelve a ensamblar el texto en una cadena limpia. Este proceso transforma el texto original en una versión más ordenada ideal para alimentar los modelos.

4.3 Prueba de la función de procesado

En este punto se evalúa la función implementada anteriormente, *preprocess_text*. Esta prueba es importante, ya que permite comprobar si la función procesa las reseñas de manera adecuada. Para ello, se aplica la función a una muestra representativa del conjunto de datos y se revisan los resultados. Se verifica, por ejemplo, que los acentos y caracteres especiales se hayan eliminado, que las palabras hayan sido normalizadas correctamente, y que se hayan eliminado las palabras muy comunes que podrían sesgar el análisis. En la Figura 36 se observa un ejemplo de cómo se realiza esta verificación.

```python
# Definiendo los códigos de color
BLACK = '\033[30m'
RED = '\033[31m'
RESET = '\033[0m'
WHITE = '\033[37m'

# Probando la función con un ejemplo
sample_text = "I love running in the park! This is a great experience."
preprocessed_sample = preprocess_text(sample_text)

print(f"{WHITE}sample_text: {sample_text}{RESET}")
print(f"{RED}preprocessed_sample: {preprocessed_sample}{RESET}")
```

```
sample_text: I love running in the park! This is a great experience.
preprocessed_sample: love running park great experience
```

Figura 36

Fragmento de código para realizar una prueba de la función preprocess_text

En la Figura 36, como se puede observar, se está probando la función *preprocess_text* con el enunciado: "*I love running in the park! This is a great experience*". Se muestra el texto original (en blanco) y el resultado después de aplicar la función (en rojo). Esto permite comprobar visualmente que la función está procesando el texto de manera adecuada y está lista para aplicarse a todo el conjunto de datos y cada una de las reseñas.

4.4 Aplicación de la función de procesado al conjunto de datos

En esta sección se describe la aplicación de la función *preprocess_text* a todo el conjunto de datos del *dataframe*. Esta fase se encarga de limpiar y estandarizar el texto de forma uniforme. La función se emplea para procesar cada entrada del conjunto de datos. Durante este proceso, se revisa cómo se transforma cada registro para asegurar que se realicen de manera constante operaciones como normalización, *tokenización*, limpieza y lematización. Esta consistencia ayuda a mantener la integridad de los datos y lograr datos confiables. Además, se presta atención al rendimiento del sistema, ya que procesar un gran volumen de datos requiere tiempo y recursos, por lo que es importante que la función sea eficaz.

```python
labeled_df['processed_reviews'] = labeled_df['reviewText'].apply(preprocess_text)
```

Figura 37

Fragmento de código para aplicar la función preprocess_text a la columna 'reviewText'

En la Figura 37 se muestra un ejemplo de cómo se está aplicando la función *preprocess_text* a cada reseña de la columna *reviewText*. Con este paso se busca transformar el texto original de cada reseña a una versión más limpia y uniforme. Además, al agregar el texto procesado como una columna adicional, se facilita la comparación con el texto original y se establece una base sólida para posteriores etapas del análisis.

```python
import random

# Obtener 10 índices aleatorios únicos dentro del rango de la longitud de tus datos
random_indices = random.sample(range(len(labeled_df)), 10)

# Iterar sobre los índices aleatorios y mostrar los elementos correspondientes
for i, index in enumerate(random_indices, 1):
    original_review = labeled_df['reviewText'].values[index]
    processed_review = labeled_df['processed_reviews'].values[index]
    print(f"{WHITE}Review original {i}: {original_review}{RESET}")
    print(f"{RED}Review procesada {i}: {processed_review}{RESET}")
```

Figura 38

Fragmento de código para realizar una comparativa del texto original con el texto procesado

En la Figura 38 se muestra la lógica para inspeccionar los textos tanto originales como los textos ya procesados. Esto permite validar que, efectivamente, el preprocesado de texto está siendo adecuado y, en caso se detecte alguna anomalía, realizar los ajustes correspondientes para obtener los mejores resultados.

```
Review original 1: I am 31 years old and I purchased this product in the summer of 2016. I like the consistency of this night cream--it's a thicker consistency than the Baxter of Cali
Review procesada 1: thirty-one year old purchased product summer like consistency night cream thicker consistency baxter california oil free moisturizer little dab cover entire face p
Review original 2: I love CK one, but the deodorant is horrible, Sicky, wet & can barely smell the CK One, was really bummed
Review procesada 2: love ck one deodorant horrible sicky wet barely smell ck one really bummed
Review original 3: This may be a great product by I couldn't get past the perfume. It is very strongly scented. I'm a little sensitive to fragrance and I knew it would bother me if i
Sorry, but I couldn't even try it.
Review procesada 3: may great product could get past perfume strongly scented little sensitive fragrance knew would bother used sorry could even try
Review original 4: Bought a week ago, used it 4 times. It worked perfectly the first couple times I used it. The last time it fried my hair literally leaving it crinkled and smelling
Review procesada 4: bought week ago used four time worked perfectly first couple time used last time fried hair literally leaving crinkled smelling like hair caught fire wo even get w
Review original 5: My wifes review since she is the one using it:  This is my first experience with a La Roche Mousse Matte Foundation product. I have oily, yet sensitive skin. I have

When I applied the La Roche foundation on my skin, it felt very light  not exactly like it was a foundation. It mostly covered my acne problems, which were not at all good at the time

From the description, I thought this foundation would work well for me.  I am from the Philippines, but I have comparatively light complexion skin  but not entirely white, either.  Th

I still wanted to try it to determine if it was suitable for me. Very recently, I received the Roche Posay Effaclar Purifying Foaming Gel Cleanser and I like it a lot  especially sinc

<a data-hook="product-link-linked" class="a-link-normal" href="/La-Roche-Posay-Effaclar-Purifying-Foaming-Gel-Cleanser-for-Oily-Skin/dp/B0033JSEGNC/ref=cm_cr_arp_d_rvw_txt?ie=UTF8">La

Ive considered buying a different shade of this foundation, but its a bit expensive. Also, the antibiotics and prescription nighttime creams my new dermatologist prescribed have virtu
Review procesada 5: wife review since one using first experience la roche mousse matte foundation product oily yet sensitive skin periodic problem adult acne thats often difficult cov
Review original 6: I wish it was a mineral sunscreen but the consistency makes me prefer it over mineral!
Review procesada 6: wish mineral sunscreen consistency make prefer mineral
Review original 7: Uneven shave.
Review procesada 7: uneven shave
Review original 8: ok have kept for my collection however still find myself using my Sonderheim butterfly opening for ease and smoothness
Review procesada 8: ok kept collection however still find using sonderheim butterfly opening ease smoothness
```

Figura 39

Resultados del procesamiento de texto

En la Figura 39 se ve la comparación visual en la que se muestran, lado a lado, algunas reseñas originales y sus versiones después de aplicar la función *preprocess_text*. Para ello, se han seleccionado diez registros de manera aleatoria y se está imprimiendo el texto antes y después del procesamiento empleando colores distintos para distinguirlos fácilmente. Esta comparación demuestra cómo la función limpia el texto.

4.5 Visualización de N-Grams en el análisis de datos procesados

En esta sección se muestran gráficos de *N-Grams* (unigramas, bigramas y trigramas) las cuales se visualizaron anteriormente en el punto 3.7 pero con la diferencia que ahora estas representaciones gráficas consideran textos ya procesados para notar la importancia de una buena limpieza o preprocesado de texto. Esto va a permitir identificar patrones de lenguaje relacionados con sentimientos, mostrando cómo se agrupan las palabras en función de si expresan una percepción positiva o negativa. Cabe mencionar que realizar el análisis de *N-Grams* permite verificar también que el proceso de limpieza de datos haya sido efectivo.

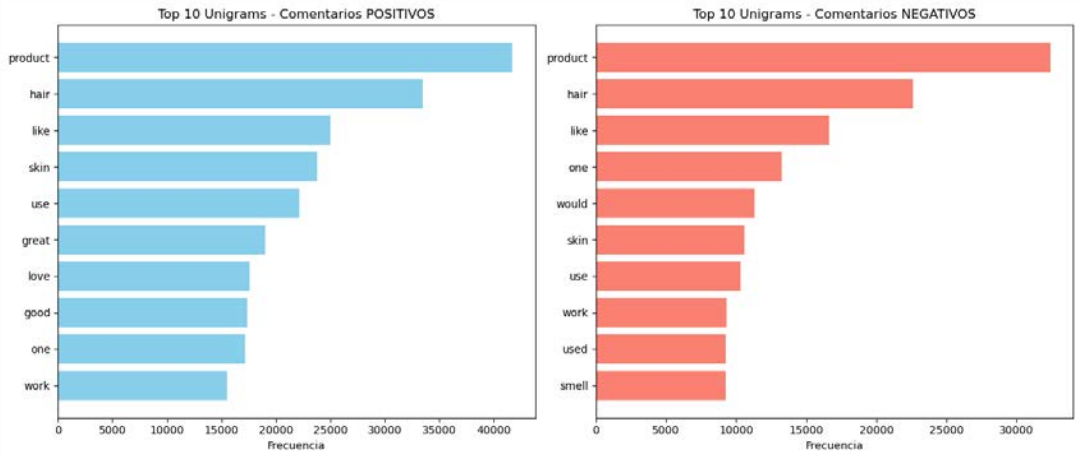


Figura 40 Unigramas para comentarios positivos y negativos

En la Figura 40 se muestran los diez unigramas más frecuentes tanto para comentarios positivos como negativos. Así mismo, como se puede ver, en los comentarios positivos se destacan palabras como "*product*", "*hair*", "*like*", "*use*", entre otros. Estos términos indican que los consumidores resaltan la efectividad del producto, muestran satisfacción. Por otro lado, en los comentarios negativos aparecen palabras como "*skin*", "*smell*", "*product*", entre otros. Aunque algunos términos son comunes a ambos grupos, en los comentarios negativos destacan palabras que expresa insatisfacción o dudas, como "*would*", "*used*" y "*smell*". Esto sugiere que los consumidores cuestionan la efectividad del producto y tienen problemas con aspectos sensoriales, como el olor, lo que relaciona con experiencias negativas o expectativas no cumplidas en productos de belleza y cuidado personal.

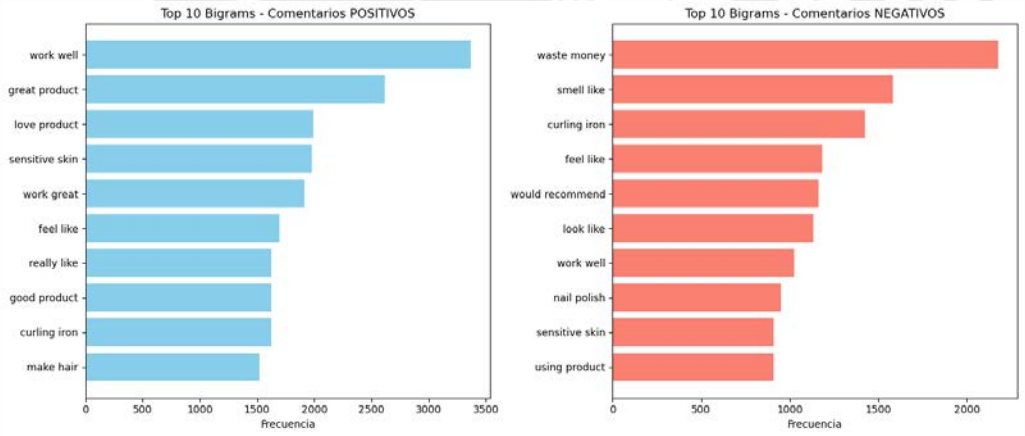


Figura 41 Bigramas para comentarios positivos y negativos

En la Figura 41 se muestran los diez bigramas más comunes en los comentarios positivos y negativos. En el gráfico de los comentarios positivos se pueden ver frases como "*work well"*, "*great product*", "*love product*", "*sensitive skin*", entre otros. Estas combinaciones indican que los consumidores resaltan la efectividad de los productos y valoran características específicas, como la adecuación para piel sensible, por ejemplo. En el gráfico de los comentarios negativos, se destacan bigramas como "*waste money*", "*smell like*", "*curling iron*", entre otros. Por ejemplo, "*waste money*" sugiere que los consumidores se sienten insatisfechos con la relación calidad-precio, mientras que "*smell like*" indica que el olor del producto es un aspecto que genera descontento, generalmente. Otras combinaciones, como "*would recommend*" y "*look like*", reflejan frustración sobre la efectividad del producto o expectativas que no se han cumplido.

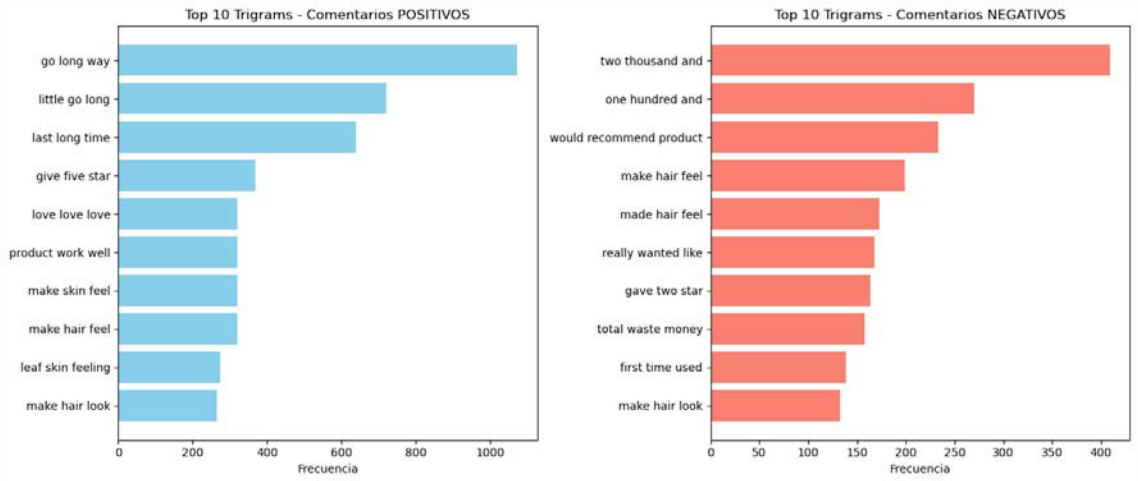


Figura 42 Trigramas para comentarios positivos y negativos

En la Figura 42 se muestran los diez trigramas más frecuentes en los comentarios positivos y negativos. En estas combinaciones queda mucho más claro la posición del consumidor frente a los productos. Este análisis de trigramas permite ver claramente que aspectos destacan los consumidores.

4.6 Almacenamiento del conjunto de datos procesados

En esta sección se describe el último paso del preprocesamiento del conjunto de datos: guardar el *dataframe* ya procesado. Una vez que la función *preprocess_text* ha transformado las reseñas, el texto original se ha convertido en una versión limpia y lista para ser utilizada en análisis de NLP y modelos de aprendizaje automático. Guardar este *dataframe* es importante, ya que permite conservar el procesado realizado a la fuente original evitando que se tenga que repetir los pasos anteriores. Además, al almacenar los datos en un formato como csv o *parquet* (dependiendo del volumen), se facilita el acceso y la colaboración entre miembros del equipo, o incluso para ser utilizado en proyectos posteriores. Cabe mencionar que este paso no solo es una buena práctica en la gestión de datos, sino que también garantiza la reproductibilidad de los resultados, permitiendo que otros investigadores o analistas verifiquen y construyan sobre el trabajo ya realizado. En la Figura 43 se muestra cómo se procede a guardar estos resultados parciales para su posterior uso en el modelado de modelos de aprendizaje automático o NLP.

```
1  labeled_df.to_csv('process_labeled_df.csv', index=False, mode='w')
```

Figura 43

Fragmento de código para almacenar el dataframe procesado

CAPÍTULO 5 – MODELOS DE MACHINE LEARNING Y DEEPLEARNING PARA ANÁLISIS DE SENTIMIENTO

Para analizar los sentimientos en el sector de lujo, se utilizan varios modelos de *machine learning*, desde métodos sencillos hasta técnicas avanzadas de *deep learning*, y se elige el más adecuado según las necesidades del proyecto. Por ejemplo, se comienza con la regresión logística, un modelo básico que es fácil de implementar y funciona bien para clasificar datos en dos categorías. Luego se prueban modelos más robustos como *Support Vector Machines* (*SVM*) y *Random Forest*. *SVM* es útil para separar claramente las clases cuando hay muchas características, mientras que *Random Forest*, que combina varios árboles de decisión, ayuda a mejorar la precisión y manejar datos complejos sin caer en el sobreajuste. También se prueba con *XGBoost* un modelo muy eficiente para clasificar especialmente cuando los datos son variados o tienen ruido y con técnicas como *GridSearch* se afinan sus parámetros para obtener el mejor rendimiento.

En el área de *deep learning*, se comienza con redes neuronales simples y luego se avanza hacia modelos más complejos. Se utilizan redes como *LSTM* y *GRU*, que son muy buenas para entender el contexto y las emociones en secuencias de texto, lo que es esencial para captar la complejidad de las reseñas.

En este capítulo se explica cómo se entrenan, ajustan y evalúan estos modelos, comparando su eficacia en el análisis de sentimientos. Por último, como se busca determinar el mejor modelo con mayor eficacia y rendimiento computacional, por lo cual se presenta una tabla comparativa con los resultados parciales de cada modelo.

5.1 Carga y visualización de datos limpios para el entrenamiento de modelos

```
import pandas as pd


processed_df = pd.read_csv('process_labeled_df.csv')

processed_df.head()
```

Figura 44

Fragmento de código para almacenar el dataframe procesado

En la Figura 44 se ve cómo se cargan y visualizan los datos desde el archivo en formato csv guardado anteriormente en el punto 4.6. Este proceso no solo va a permite confirmar que la información previamente procesada se ha guardado correctamente, sino que también ayuda a verificar que los datos estén en el formato adecuado para realizar el modelo en caso contrario se debe regresar al paso anterior y volver a procesar toda la información. En la Figura 45 se detalla la descripción de cada campo presente en el *dataframe*.

	overall	reviewText	review_length	label	processed_reviews
0	5.0	Great product . Life saver for people who stru...	67	1	great product life saver people struggle break
1	5.0	World great !	13	1	world great
2	5.0	ultra compact and works very well	33	1	ultra compact work well
3	5.0	It's is my long-time favorite and I'm not trad...	442	1	favorite trading shampoo long work way leaf ha...
4	5.0	This is a very light but masculine, slightly c...	131	1	light masculine slightly citrus cologne approp...

- **overall:** Representa la calificación dada por el usuario al producto.
- **reviewText:** La revisión original proporcionada por el usuario.
- **review_length:** La longitud de la revisión original.
- **label:** Representa si la revisión es positiva (1) o negativa (0).
- **processed_reviews:** La revisión después de aplicar el preprocesamiento.

Figura 45

Descripción de los campos del dataframe procesado

5.2 División del conjunto de datos en entrenamiento y prueba

La división del conjunto de datos en partes de entrenamiento y prueba es común para desarrollar modelos de *machine learning* y *deep learning*. Este paso permite evitar que el modelo se "aprenda" demasiado los detalles específicos del conjunto de entrenamiento, lo que se conoce como sobreajuste, y permite evaluar de forma realista su desempeño. En este caso, se usa la función *train_test_split* de *Scikit-Learn* para separar el conjunto de datos en dos partes: el 70% se utiliza para entrenar el modelo y el 30% se reserva para probarlo. La elección de estos porcentajes busca un equilibrio adecuado entre tener suficientes datos para que el modelo aprenda y contar con un conjunto de prueba representativo para evaluar su rendimiento. La Figura 46 muestra el código que realiza esta división.

```
# Dividir los datos en conjuntos de entrenamiento y prueba (70% entrenamiento, 30% prueba)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Figura 46

Fragmento de código para división del conjunto de datos en entrenamiento y prueba

5.3 Definición de métricas para la evaluación de modelos

La evaluación detallada de los modelos de *machine learning* y *deep learning* es importante para garantizar su efectividad en el análisis de sentimientos. Por ello, se van a emplear las métricas explicadas en el punto 2.4.6. En conjunto, estas métricas no solo permiten medir la efectividad del modelo, sino que también guían los ajustes necesarios para mejorar su desempeño.

5.4 Modelo: Regresión Logística

La regresión logística se emplea para predecir si algo pertenece a una categoría u otra, como determinar si un mensaje es spam o no. Aunque su nombre pude sonar similar a la regresión lineal, este modelo se emplea específicamente para clasificación, no para predecir valores continuos. Funciona asignando a cada entrada un valor de cero y uno mediante una función sigmoide, que se interpreta como la probabilidad de que esa entrada pertenezca a una de las clases. Esto la hace adecuada para problemas donde solo hay dos posibles resultados como este caso.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
```

Figura 47

Fragmento de código para importar clases de "Scikit-learn" y el modelo de regresión logística

En la Figura 47 se describen algunas de las herramientas clave que se emplean para realizar el modelado. Por ejemplo, el *TfidVectorized* se encarga de transformar textos en números, de modo que se pueda medir qué tan relevante es cada palabra en un documento en comparación con todo el conjunto de documentos. Esto se hace contando cuántas veces aparece una palabra en un texto y ajustando ese valor según la rareza de la palabra en el corpus. Otra herramienta es *train_test_split*, que ya se explicó anteriormente, pero se encarga de dividir el conjunto de datos en entrenamiento y pruebas. El modelo *LogisticRegression* se utiliza para clasificar datos, es decir, para predecir a qué categoría pertenece cada ejemplo. Este modelo utiliza una función que convierte los resultados en probabilidades, siendo especialmente útil cuando hay dos opciones, aunque también se puede adaptar a situaciones con más categorías. También se emplea *classification_report* y *accuracy_score* para evaluar el desempeño del modelo. La primera ofrece un resumen con métricas que indican la precisión y el rendimiento para cada clase, mientras que la segunda mide la proporción de predicciones correctas.

```
# Vectorización de las reviews usando TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=5000)  # Limitamos a 5000 características para evitar vectores demasiado grandes
X = tfidf_vectorizer.fit_transform(processed_df['processed_reviews'].astype(str))
y = processed_df['label']
```

Figura 48

Fragmento de código para ejecutar la vectorización

En la Figura 48 se puede ver cómo se emplea el *TfidfVectorizer* para procesar texto. Al inicializar el vectorizador con *max_features* igual a 5000, se limita el número de palabras a las 5000 más importantes, lo que reduce la complejidad de los datos y ayuda a evitar que el modelo se ajuste demasiado a los detalles específicos esta elección es en base a la experiencia y, que en muchos casos y otros contextos puede variar. Luego la función *fit_transform* se encarga de ajustar el vectorizador al conjunto de datos y convertir estos textos en una matriz numérica (la matriz *TF-IDF*) en una sola operación. En esta matriz, cada fila representa un documento y cada columna una palabra, y los valores indican la importancia de cada palabra en cada documento. Por otro lado, "y" es el vector que tiene las etiquetas de cada reseña, las cuales se emplean para entrenar el modelo.

```
# Dividir los datos en conjuntos de entrenamiento y prueba (70% entrenamiento, 30% prueba)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Figura 49

Fragmento de código para dividir el conjunto de datos en entrenamiento y prueba

En la Figura 49 se ve cómo se divide el conjunto de datos usando la función *train_test_split*. Con el 30% de los datos reservados para probar el modelo, mientras que el 70% restantes se utiliza para entrenar el modelo. Además, se emplea un *random_state* igual 42 para que la división sea siempre la misma cada vez que se ejecuta el código.

Figura 50

Fragmento de código para ejecutar el entrenamiento del modelo de regresión logística

En la Figura 50, se observa cómo se inicializa el modelo con *LogisticReression*, luego, mediante *fit(X_train, y_train),* el modelo se ajusta a los datos de entrenamiento.

Figura 51

Fragmento de código para ejecutar la predicción usando el modelo de regresión logística

En la Figura 51 se observa como la función *predict(X_test)* aplica el modelo que se entrenó previamente a los datos de prueba para terminar a qué categoría pertenecen. Esto ayuda a verificar cómo de bien el modelo funciona con información que no ha visto antes.

Figura 52

Evaluación del modelo de regresión logística

En la Figura 52 se muestras las funciones utilizadas para evaluar el rendimiento del modelo de regresión logística. Por un lado, la función *accuracy_score(y_text, y_pred)* calcula el porcentaje de predicciones correctas, lo cual es útil para tener una idea general del desempeño del modelo. Sin embargo, cabe mencionar, que, en conjuntos de datos desequilibrados, ésta métrica puede no reflejar bien cómo se comporta el modelo con las clases menos frecuentes. Por ello se utiliza también la función *classification_report(y_test, y_pred),* que ofrece un reporte más detallado incluyendo métricas como precisión, *recall* y F1-score para cada clase.

```
Accuracy (70-30 split): 0.8012

Classification Report (70-30 split):
              precision    recall  f1-score   support

           0       0.77      0.71      0.74     17685
           1       0.82      0.86      0.84     26568

    accuracy                           0.80     44253
   macro avg       0.80      0.79      0.79     44253
weighted avg       0.80      0.80      0.80     44253
```

Figura 53

Resultados del modelo de regresión logística

En la Figura 53 se presentan los resultados de evaluación del modelo de regresión logística en el conjunto de prueba, utilizando varias métricas de rendimiento clave: precisión, *recall*, F1-score y *support*. En este caso, de las métricas presentadas, la que tiene más peso al momento de compararlos con otros modelos es el F1-Score ya que esta métrica combina, como se mencionó, tanto la precisión y el *recall*. Por ello, como se ve en la Figura 53 el F1-Score tiene un promedio ponderado de 0.80. Esto indica que, en general, el modelo tiene un buen equilibrio y que el modelo puede ser confiable para el análisis de sentimiento en este contexto.

5.5 Modelo: Máquinas de Vector Soporte (SVM)

Las *Support Vector Machines* (SVM) buscan encontrar la mejor "línea" o separación en el espacio de características que distinga entre dos grupos de datos. La idea principal es maximizar el margen, es decir, lograr que la distancia entre esta línea y los puntos más cercanos de cada grupo (los llamados vectores soporte) sea lo mayor posible. Este enfoque mejora la capacidad del modelo para generalizar, lo que significa que funcionará mejor al clasificar datos nuevos que no se ha visto antes.

```
1  from sklearn.svm import LinearSVC
```

Figura 54

Fragmento de código para importar el modelo SVM

En la Figura 54 se importa el modelo *LinearSVC*, que es una implementación de *Support Vector Machine* (*SVM*) para la clasificación lineal.

Como se puede notar, ya no es necesario realizar el preprocesamiento de los conjuntos de entrenamiento y prueba en este punto. Este paso inicial se llevó a cabo al inicio del análisis asegurando que los datos estuvieran listos para ser utilizados en varios modelos. Ahora, el enfoque se centra en seleccionar el modelo adecuado para un análisis más detallado y comparativo.

```python
# Entrenando el modelo SVM
svm_model = LinearSVC(random_state=42, max_iter=10000)
svm_model.fit(X_train, y_train)
```

Figura 55

Fragmento de código para ejecutar el entrenamiento del modelo de SVM

En la Figura 55 se describen los parámetros y el proceso de ajuste del modelo *SVM* con *LinearSVC*. Por ejemplo, como en el caso anterior, se tiene *random_state* igual 42 para mantener la distribución de los datos. Además, este modelo tiene un parámetro adicional *max_iter* el cual, para este caso, se establece en 10000, que va a definir el número máximo de iteraciones que el algoritmo realizará para alcanzar una solución estable. Este parámetro es útil cuando se trabaja con conjunto de datos grandes o complejos, ya que permite que el algoritmo tenga suficiente tiempo para converger como en el caso anterior es una variable que puede variar dependiendo del contexto y los resultados que se obtengan en cada ensayo.

```python
# Predicciones en el conjunto de prueba
y_pred_svm = svm_model.predict(X_test)
```

Figura 56

Fragmento de código para ejecutar la predicción usando el modelo de SVM

En la Figura 56 se describe el proceso de predicción utilizando el modelo SVM entrenado al igual que en el modelo anterior.

```python
# Evaluación del modelo
accuracy_svm = accuracy_score(y_test, y_pred_svm)
classification_rep_svm = classification_report(y_test, y_pred_svm)
```

Figura 57

Evaluación del modelo de SVM

En la Figura 54 se muestras las funciones utilizadas para evaluar el rendimiento del modelo de *SVM*. El uso de un modelo *SVM* lineal (*LinearSVC*) es especialmente eficaz cuando las clases se pueden separar claramente en el espacio de características. Este modelo, en particular, se beneficia de la representación del texto mediante *TF-IDF*, una técnica, como se mencionó anteriormente, que convierte el texto en números basados en la frecuencia y relevancia de las palabras, permitiendo manejar grandes volúmenes de datos de forma eficiente. Anteriormente, se usó *Word2Vec* para explorar las relaciones entre palabras, ya que genera representaciones densas y captura el contexto. No obstante, para la tarea de clasificación, *TD-IDF* resulta más apropiado debido a que estructura el texto de manera organizada y facilita una separación entre clases. A continuación, se analizan los resultados obtenidos por este modelo, los cuales se describen en la Figura 58.

```
Accuracy (70-30 split): 0.7978

Classification Report (70-30 split):
              precision    recall  f1-score   support

           0       0.76      0.71      0.74     17685
           1       0.82      0.85      0.84     26568

    accuracy                           0.80     44253
   macro avg       0.79      0.78      0.79     44253
weighted avg       0.80      0.80      0.80     44253
```

Figura 58

Resultados del modelo de SVM

Como se ve en el gráfico, centrándose en el F1-Score, que es la métrica clave para evaluar el equilibrio entre la precisión y el *recall*. Para la clase 0 (o clase negativa), se tiene un F1-Score de 0.74, aunque el modelo identifica una buena parte de las instancias de esta clase, todavía hay margen para mejorar su capacidad de captura. En la clase 1, el F1-Score es de 0.84, lo que indica un desempeño robusto en la clasificación de estas instancias. Como se ve en la Figura 58 el F1-Score tiene un promedio ponderado de 0.80. Esto indica que, en general, el modelo tiene un buen equilibrio y que el modelo puede ser confiable para el análisis de sentimiento en este contexto.

5.6 Modelo: Random Forest

Las *Random Forest* trabajan construyendo varios árboles de decisión, cada uno entrenado con una parte aleatoria del conjunto de datos. Para clasificar, cada árbol "vota" por una clase y la decisión final se toma según la mayoría de los votos. Este método mejora la precisión y reduce el riesgo de sobreajuste, ya que aprovecha la diversidad de los árboles, lo que resulta en un modelo más robusto y capaz de generalizar mejor nuevos datos.

```
1  from sklearn.ensemble import RandomForestClassifier
```

Figura 59

Fragmento de código para importar el modelo random forest

En la Figura 59 se presenta la clase *RandomForestClassifier*, que es la implementación del algoritmo de clasificación *Random Forest* en la biblioteca *Scikit-learn.*

```
1  # Entrenando el modelo de Bosques Aleatorios
2  rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
3  rf_model.fit(X_train, y_train)
4
5  # Predicciones en el conjunto de prueba
6  y_pred_rf = rf_model.predict(X_test)
```

Figura 60

Fragmento de código para ejecutar el entrenamiento del modelo de random forest

En la Figura 60 se muestra cómo se configura un modelo *RandomForestClassifier*. En particular, el parámetro *n_estimator* igual a 100 es importante, ya que indica que el modelo está formado por 100 árboles al igual que en los casos anteriores esta decisión se basa en la experiencia y los resultados. Además, se debe mencionar que tener un número mayor de árboles permite capturar más patrones en los datos, lo que generalmente mejora la precisión del modelo. El resto de los parámetros se mantiene como en los modelos anteriores.

```python
# Evaluación del modelo
accuracy_rf = accuracy_score(y_test, y_pred_rf)
classification_rep_rf = classification_report(y_test, y_pred_rf)
```

Figura 61

Evaluación del modelo de random forest

En la Figura 61 se muestra cómo se generan los reportes de métricas como se ha visto en los modelos anteriores. He de mencionar que *Random Forest* es muy valorado en *machine learning* por su robustez, buena precisión y capacidad para manejar datos con muchas características, incluso cuando hay ruido o datos faltantes. Esto lo hace igual para trabajar con grandes volúmenes de información y problemas complejos.

```
Accuracy (70-30 split): 0.8067

Classification Report (70-30 split):
              precision    recall  f1-score   support

           0       0.80      0.69      0.74     17685
           1       0.81      0.88      0.85     26568

    accuracy                           0.81     44253
   macro avg       0.80      0.79      0.79     44253
weighted avg       0.81      0.81      0.80     44253
```

Figura 62

Resultados del modelo de random forest

En la Figura 62 se muestra que, para el modelo *Random Forest*, la clase 0 tiene un F1-Score de 0.74, lo que indica un equilibrio moderado entre la capacidad de predecir correctamente y la de capturar todas las instancias reales. En cambio, para la clase 1, el F1-Score es de 0.85, lo que refleja un rendimiento robusto y buena capacidad para identificar correctamente las instancias de esta categoría.

5.7 Modelo: Gradient Boosting (XGBoost)

Gradient Boosting construye un modelo predictivo combinando varios modelos más sencillos, generalmente árboles de decisión. Este proceso es iterativo: cada árbol nuevo se enfoca en corregir los errores realizados por los árboles previos, lo que mejora gradualmente la precisión del modelo en casa paso.

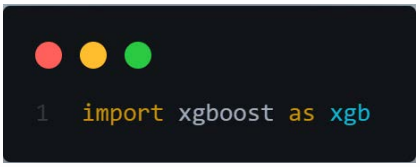


Figura 63

Fragmento de código para importar el XGBoost

En la Figura 63 se presenta la importación del modelo *XGBoost*, una biblioteca optimizada para el algoritmo de *Gradient Boosting*. *XGBoost* es altamente eficiente y potente, lo que lo hace muy popular en competencias de *machine learning* debido a su rendimiento superior.

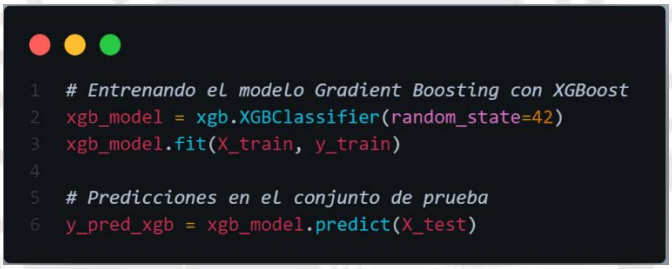


Figura 64

Fragmento de código para ejecutar el entrenamiento del modelo de XGBoost

En la Figura 64 se describen los componentes clave de la implementación de *XGBClassifier*, que es una clase para construir un clasificador utilizando el algoritmo de *Gradient Boosting* con la biblioteca *XGBoost*. Es importante destacar que, en este caso, no se han ajustado otros hiperparámetros para el modelo, aunque parámetros como el número de árboles (*n_estimators*), la profundidad máxima de los árboles (*max_depth*), la tasa de aprendizaje (*learning_rate*) y otros pueden configurarse para mejorar aún más el rendimiento. Aquí, se está optando por usar la configuración por defecto, lo que permite evaluar la efectividad del modelo sin modificaciones adicionales en sus hiperparámetros.

Figura 65

Evaluación del modelo de XGBoost

En la Figura 65 se muestra cómo se generan los reportes de métricas como se ha visto en los modelos anteriores. A continuación, se interpretarán los resultados obtenidos por el modelo *XGBoost*, los cuales se presentan en la Figura 63.

```
Accuracy (70-30 split): 0.7801

Classification Report (70-30 split):
              precision    recall  f1-score   support

           0       0.77      0.64      0.70     17685
           1       0.79      0.87      0.83     26568

    accuracy                           0.78     44253
   macro avg       0.78      0.76      0.76     44253
weighted avg       0.78      0.78      0.78     44253
```

Figura 66

Resultados del modelo de XGBoost

En la Figura 66 se muestran los resultados del modelo *XGBoost* en el conjunto de prueba, centrándonos en el F1-Score para evaluar su rendimiento. Para la clase 0, el F1-Score es de 0.70, lo que es relativamente moderado, esto sugiere que, aunque el modelo acierta en la mayoría de los casos etiquetados como clase 0, aún se pierden bastantes instancias reales. En cambio, para la clase 1, el F1-Score es de 0.83, reflejando un desempeño más robusto, aunque el modelo funciona bien en la clase 1, existe margen de mejorar la detección de la clase 0 implementando o configurando los hiperparámetros del modelo.

5.8 Modelo: CatBoost

CatBoost es un algoritmo de *boosting* que se destaca por trabajar directamente con datos categóricos sin necesidad de convertirlos en múltiples variables mediante técnicas como el "*one-hote-encoding*". Esto lo hace especialmente útil en áreas donde la mayoría de información se presenta en forma de categorías, simplificando el procesamiento y mejorando la eficiencia del modelo.

```python
import catboost as cb
```

Figura 67

Fragmento de código para importar el CatBoost

En la Figura 67, se presenta la importación de la biblioteca *CatBoost*.

```python
# Entrenando el modelo Gradient Boosting con CatBoost
cb_model = cb.CatBoostClassifier(random_state=42, verbose=0)
cb_model.fit(X_train, y_train)

# Predicciones en el conjunto de prueba
y_pred_cb = cb_model.predict(X_test)
```

Figura 68

Fragmento de código para ejecutar el entrenamiento del modelo de CatBoost

En la Figura 68 se muestra cómo se inicializa el *CatBoostClassifier*. Además, de los parámetros utilizados anteriormente, en este caso se ha configurado un "verbose" igual a cero, lo que significa que durante el entrenamiento no se muestran mensajes de progreso en la consola. Esto ayuda para mantener la salida limpia, especialmente cuando el entrenamiento es largo o se realizan muchas iteraciones lo que en ocasiones evita el análisis del entrenamiento del modelo.

```
# Evaluación del modelo
accuracy_cb = accuracy_score(y_test, y_pred_cb)
classification_rep_cb = classification_report(y_test, y_pred_cb)
```

Figura 69

Evaluación del modelo de CatBoost

En la Figura 69 se muestra cómo se generan los reportes de métricas como se ha visto en los modelos anteriores. En la Figura 70, se presentan los resultados del modelo *CatBoost*, que se interpretan a continuación.

```
Accuracy (70-30 split): 0.7989

Classification Report (70-30 split):
              precision    recall  f1-score   support

           0       0.78      0.69      0.73     17685
           1       0.81      0.87      0.84     26568

    accuracy                           0.80     44253
   macro avg       0.80      0.78      0.79     44253
weighted avg       0.80      0.80      0.80     44253
```

Figura 70

Resultados del modelo de CatBoost

Se observa que para la clase 0, el F1-Score es de 0.73 manteniendo un equilibrio moderado, aunque existe el margen para mejorar la identificación completa de esta clase. Por otra parte, en la clase 1, el F1-Score es de 0.84, reflejando un rendimiento. En definitiva, se puede mejorar la clase 0 haciendo una configuración de hiperparámetros para lograr una detección más completa.

5.9 Modelo: XGBoost - Optimizado

La optimización de modelos *XGBoost* se centra en encontrar los mejores ajustes para que el modelo funcione de manera óptima. Para lograr esto, se emplea una técnica llamada búsqueda en cuadrícula (*GridSearch*). Esta técnica evalúa de forma sistemática diferentes combinaciones de parámetros predefinidos para identificar cuál de ellas ofrece el mejor rendimiento según una métrica de evaluación específica.

```python
from sklearn.model_selection import GridSearchCV

# Definimos los hiperparámetros que queremos optimizar
param_grid = {
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5, 7],
    'n_estimators': [50, 100, 200]
}
```

Figura 71

Fragmento de código para implementar el GridSearchCV

En la Figura 71 se observa cómo se emplea el *GridSeachCV* de *Scikit-learn* para identificar de manera sistemática la mejor combinación de hiperparámetros que optimice el rendimiento del modelo. Para lograrlo, se define un diccionario, llamado *param_grid*, en el que se especifican los hiperparámetros a testear. En este caso, se han seleccionado tres parámetros claves: el *learning rate*, que determina la velocidad a la que el modelo aprende; el *max_depth*, el cual fija la profundidad máxima de los árboles y permite evaluar su influencia en el aprendizaje; y el *n_estimators*, que indica el número de árboles de decisión que van a componer el modelo. Este enfoque facilita encontrar la configuración óptima, lo que a su vez mejora el rendimiento como la capacidad de generalización.

```python
# Definimos el modelo
xgb_classifier = xgb.XGBClassifier(random_state=42)

# Establecemos la validación cruzada y la búsqueda en el espacio de hiperparámetros
grid_search = GridSearchCV(xgb_classifier, param_grid, cv=3, scoring='accuracy', n_jobs=-1, verbose=1)

# Realizamos la búsqueda
grid_search.fit(X_train, y_train)
```

Figura 72

Fragmento de código para ejecutar el entrenamiento del modelo de XGBoost optimizado

En la Figura 72 se muestra cómo se optimizan los hiperparámetros de un modelo *XGBoost* utilizando *GridSearch*, lo que permite buscar la mejor configuración para maximizar la precisión del modelo. En este proceso, el parámetro *cv* igual a tres divide el conjunto de datos en tres partes, de forma que el modelo se evalúa en diferentes subconjuntos, garantizando que el rendimiento observado sea consistente. La métrica de evaluación se establece con *scoring* igual a *acurracy*, lo que significa que se mide la proporción de predicciones correctas. Además, el uso de *n_jobs* igual a -1 permite que *GridSearchCV* aproveche todos los núcleos disponibles en el sistema, acelerando la búsqueda de la mejor combinación de parámetros. Este es un proceso que puede tardar varios minutos por que, lo que se trata, es intentar todas las posibles combinaciones de hiperparámetros y seleccionar aquella combinación que ofrece el mejor resultado.

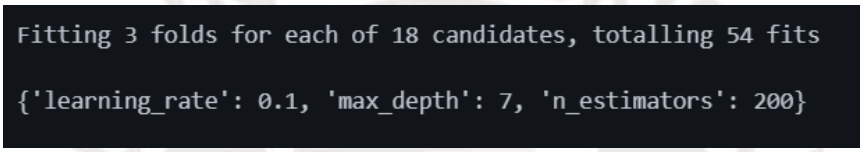
```
# Obtenemos el mejor modelo y sus hiperparámetros
best_xgb = grid_search.best_estimator_
best_params = grid_search.best_params_

best_params
```

Figura 73

Fragmento de código para obtener los mejores hiperparámetros

En la Figura 73 se muestra cómo se obtienen los valores de los hiperparámetros. El atributo *grid_seach.best_estimator_* indica el modelo que obtuvo el mejor desempeño durante la búsqueda, es decir, la combinación de parámetros que maximiza la precisión mediante validación cruzada. Por otro lado, *grid_seach.best_params_* indica cuáles fueron los valores específicos de los hiperparámetros que resultaron ser los más efectivos. Estos valores se imprimen para facilitar la replicación del modelo o su ajuste en futuros experimentos.



```
Fitting 3 folds for each of 18 candidates, totalling 54 fits

{'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200}
```

Figura 74

Valores de los mejores hiperparámetros para el modelo XGBoost optimizado

En la Figura 74 se muestra cómo se obtuvieron los valores de los hiperparámetros usando *GridSearchCV* con validación cruzada en tres particiones. Se evalúan 18 combinaciones de parámetros, lo que, multiplicado por las tres particiones, resulta un total de 54 entrenamientos. Como resultado de este proceso, se identificaron los mejores hiperparámetros. Estos valores indican que el modelo aprende a una velocidad equilibrada, con árboles que no son demasiado profundos para evitar el sobreajuste, y utiliza una cantidad suficiente de árboles para mejorar su precisión sin que el entrenamiento se vuelva excesivamente lento.

```
# Definimos y entrenamos el modelo con los hiperparámetros óptimos
optimized_xgb = xgb.XGBClassifier(
    learning_rate=0.1,
    max_depth=7,
    n_estimators=200,
    random_state=42,
    n_jobs=-1  # Usa todos los núcleos disponibles para el entrenamiento
)

optimized_xgb.fit(X_train, y_train)
```

Figura 75

Fragmento de código para ejecutar el entrenamiento del modelo de XGBoost optimizado usando los mejores hiperparámetros

En la Figura 75 se detalla el proceso de ajuste del modelo *XGBClassifier* utilizando los mejores hiperparámetros encontrados. Por ejemplo, el *learning_rate* controla la velocidad a la que el modelo actualiza sus pesos durante el entrenamiento, determinando el tamaño de los pasos que se dan al ajustar el modelo. Por otro lado, el parámetro *max_depth* limita la profundidad máxima de

los árboles de decisión, lo que es evita el sobreajuste, asegurando que el modelo no se ajuste demasiado a los datos de entrenamiento. Así mismo, el *n_estimators* define la cantidad de árboles de decisión que se entrenan en secuencia, permitiendo que cada árbol intente corregir los errores del anterior y logrando un equilibrio adecuado entre precisión y tiempo de entrenamiento. Finalmente, *n_jobs* permite aprovechar todos los núcleos del procesador para realizar el entrenamiento en paralelo, acelerando considerablemente el proceso.

```python
# Evaluamos el rendimiento en el conjunto de test
y_pred_optimized = optimized_xgb.predict(X_test)
accuracy_optimized = accuracy_score(y_test, y_pred_optimized)
classification_rep_optimized = classification_report(y_test, y_pred_optimized)
```

Figura 76

Evaluación del modelo de XGBoost optimizado

En la Figura 76 se ilustra cómo se utiliza el modelo entrenado para hacer predicciones sobre conjunto de datos y, así mismo, los resultados de ello. En la Figura 77 se presentan los resultados del modelo *XGBoost Optimizado*, que se interpretan a continuación, destacando la importancia del F1-Score.

```
Accuracy (70-30 split): 0.7780

Classification Report (70-30 split):
              precision    recall  f1-score   support

           0       0.78      0.63      0.69     17685
           1       0.78      0.88      0.83     26568

    accuracy                           0.78     44253
   macro avg       0.78      0.75      0.76     44253
weighted avg       0.78      0.78      0.77     44253
```

Figura 77

Resultados del modelo de XGBoost optimizado

En la Figura 77 se muestran los resultados del modelo *XGBoost optimizado*. En la clase 0, el F1-Score es de 0.69, lo que indica que, a pesar de una precisión razonable, el modelo podría mejorar su capacidad para detectar todas las instancias reales de esa clase. En contraste, la clase 1 presenta un F1-Score de 0.83, reflejando un buen desempeño.

5.10 Modelo: Deeplearning – Redes Neuronales Simples

Una red neuronal simple se compone de una capa de entrada, una o más capas ocultas, y una capa de salida. Cada capa está formada por neuronas artificiales (también llamadas nodos o unidades) que están interconectadas.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

Figura 78

Fragmento de código para importar modelos de 'deep learning'

En la Figura 78 se importan las librerías necesarias para implementar una red neuronal, en otras palabras, un modelo de *deep learning*. *Keras*, que funciona sobre *TensorFlow*, simplifica la tarea, permitiendo construir modelos de forma secuencial, donde la información pasa de una capa a otra en una sola dirección. En este modelo se usan capas *Dense*, que son completamente conectadas, y capas *Dropout*, que permiten evitar el sobreajuste apagando aleatoriamente algunas neuronas durante el entrenamiento. Además, se utiliza el optimizador *Adam*, que ajusta automáticamente la tasa de aprendizaje para mejorar la convergencia del modelo. Así mismo, se incorpora *callback EarlyStopping*, que detiene el entrenamiento cuando la mejora en la pérdida de validación se detiene, lo que protege el sobreajuste y garantiza un entrenamiento más eficiente.

```python
# Convertir matrices dispersas a densas
X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()
```

Figura 79

Fragmento de código para convertir de matrices dispersas a densas

En la Figura 79 se muestra cómo se convierten las matrices dispersas en matrices densas. Las matrices dispersas son útiles para ahorrar memoria cuando los datos contienen muchos ceros, pero modelos de *deep learning* necesitan trabajar con matrices densas. Por eso, en este paso se transforma tanto la matriz de entrenamiento como la de prueba a un formato denso utilizando el método *toarray()*.

```python
# Crear el modelo
model = Sequential()
model.add(Dense(256, input_shape=(X_train_dense.shape[1],), activation='relu'))
model.add(Dropout(0.3))  # Ajusta La tasa de dropout
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
```

Figura 80

Fragmento de código para crear el modelo de red neuronal simple

En la Figura 80 se presenta la arquitectura del modelo de red neuronal, construido utilizando la clase *Sequential*, que permite apilar las capas de forma secuencial. El modelo comienza con una primera capa densa que tiene 256 neuronas y utiliza la función de activación *ReLu*, muy común en el campo del *deep learning*, por su sencillez y capacidad para introducir no linealidades. Para evitar que el modelo se ajuste demasiado a los datos, se aplica un *dropout* del 30% en esta capa, lo que significa que, durante cada paso de entrenamiento, el 30% de las neuronas se desactivan de forma aleatoria. La segunda capa es también densa, con 128 neuronas y la misma función de activación *ReLu*, acompañada de otro *dropout* del 30%. Por último, la capa de salida consta de una única neurona con activación *sigmoid*, ideal para problemas de clasificación binaria, ya que su salida se interpreta como una probabilidad entre cero y uno.

```
# Optimizador con tasa de aprendizaje ajustable
optimizer = Adam(learning_rate=0.001)  # Prueba con diferentes valores aquí

# Compilar el modelo
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

Figura 81

Fragmento de código optimizar y compilar el modelo de red neuronal simple

En la Figura 81 se muestra cómo se optimiza y compila el modelo. Primero se utiliza el optimizador "Adam" con una tasa de aprendizaje de 0.001; este optimizador ajusta de forma inteligente la velocidad de aprendizaje basándose en los gradientes, lo que ayuda a que el modelo aprenda de manera más eficiente. Luego, el modelo se compila usando la función de pérdida *binary_crossentropy*, adecuada para problemas de clasificación binaria, y se establece la métrica *accuracy* para evaluar el rendimiento durante el entrenamiento y la validación.

```
# Early Stopping para prevenir sobreajuste
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

Figura 82

Fragmento de código para implementar el 'early stopping'

En la Figura 82 se muestra el uso de *callback EarlyStopping*. Este mecanismo detiene el entrenamiento si la pérdida de validación no mejora tras cinco épocas y, además, restaura los pesos del modelo a los de la época en la que se obtiene el mejor rendimiento. Esto ayuda a prevenir el sobreajuste y a reducir el tiempo de entrenamiento, ya que se evita continuar el proceso cuando ya no se observan mejoras significativas.

```
# Entrenar el modelo
history = model.fit(X_train_dense, y_train, epochs=20, batch_size=64,
                    validation_data=(X_test_dense, y_test), callbacks=[early_stopping])
```

Figura 83

Entrenamiento del modelo de red neuronal simple

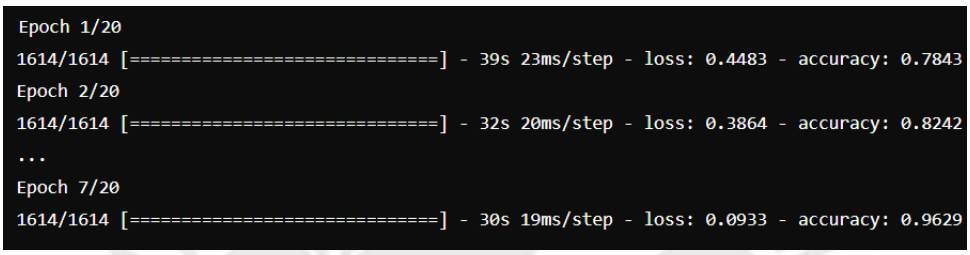
En la Figura 83 se muestra el proceso de entrenamiento del modelo. Se establece que el modelo se entrene durante veinte épocas, lo que significa que el proceso de aprendizaje se repite veinte veces sobre todo el conjunto de datos.

```
# Evaluar el modelo
accuracy = model.evaluate(X_test_dense, y_test)[1]
print(f"Accuracy: {accuracy:.4f}")
```

Figura 84

Evaluación del modelo de red neuronal simple

En la Figura 84 se ve cómo se evalúa el rendimiento del modelo utilizando los datos de prueba. En este proceso, se calcula la precisión. El modelo en cuestión es una red neuronal profunda creada y entrenada con *TensorFlow* y *Keras*. En la Figura 85 se muestran los resultados obtenidos por el modelo para cada época, lo que permite visualizar la evolución de su rendimiento durante el entrenamiento.



```
Epoch 1/20
1614/1614 [==============================] - 39s 23ms/step - loss: 0.4483 - accuracy: 0.7843
Epoch 2/20
1614/1614 [==============================] - 32s 20ms/step - loss: 0.3864 - accuracy: 0.8242
...
Epoch 7/20
1614/1614 [==============================] - 30s 19ms/step - loss: 0.0933 - accuracy: 0.9629
```

Figura 85

Evaluación del modelo de red neuronal simple para cada 'epoch'

En la Figura 85 se analiza el desempeño del modelo durante su entrenamiento. Aunque inicialmente se planificaron veinte épocas, el entrenamiento se detuvo en la séptima, ya que se activó el *callback EarlyStopping* por la falta de mejoras en la pérdida de validación durante cinco épocas consecutivas. Esto indica que el modelo dejó de mejorar y se previno el sobreajuste al restaurar los mejores pesos alcanzados.

```
1383/1383 [==============================] - 4s 3ms/step - loss: 0.4178 - accuracy: 0.8076
```

Figura 86

Evaluación final del modelo de red neuronal simple

En la Figura 83 se ve la evaluación final del modelo. En el conjunto de prueba, el modelo alcanza una pérdida de 0.4178 y una precisión del 80.76%, lo que coincide con los resultados de validación observados durante el entrenamiento. Sin embargo, se detectan algunas señales de sobreajuste, ya que la pérdida de validación (*val_loss*) aumenta en las últimas épocas. Aunque el uso de técnicas como el *Dropout* y el *callback EarlyStopping* ha ayudado a mitigar este problema, los indicadores fluctuantes en *val_loss* y *val_accuracy* sugieren que aún podría haber margen para mejorar la capacidad de generalización del modelo.

5.11 Modelo: Deeplearning – LSTM (Long Short-Term Memory)

LSTM, o *Long Short-Term Memory*, es un tipo de red neuronal recurrente diseñada específicamente para trabajar con secuencias y capturar dependencias a lo largo del tiempo. Las redes *LSTM* introducen celdas de memoria, que permiten retener información durante periodos prolongados, facilitando así el aprendizaje y la identificación de patrones incluso cuando los datos relevantes están separados por largos intervalos de tiempo.

```
print(processed_df['processed_reviews'].isnull().sum())
processed_df = processed_df.dropna(subset=['processed_reviews'])
processed_df['processed_reviews'].fillna("", inplace=True)
```

Figura 87

Fragmento de código para acondiciona los datos al modelo LSTM

En la Figura 87 se muestra cómo se gestiona la presencia de datos faltantes en la columna *processed_reviews*. El objetivo es identificar y solucionar los valores nulos, asegurando que la columna esté completa. Primero se cuenta cuántos datos faltan, luego se eliminan las filas que contienen estos valores nulos, y finalmente, se rellenan los que puedan haber quedado con una cadena vacía para garantizar la integridad del conjunto de datos.

```
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import pad_sequences

MAX_NUM_WORDS = 100
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(processed_df['processed_reviews'])
sequences = tokenizer.texts_to_sequences(processed_df['processed_reviews'])
```

Figura 88

Fragmento de código para acondiciona los datos al modelo LSTM considerando una tokenizacion

En la Figura 88 se observa el proceso de transformar los textos para prepararlos para una red *LSTM*. En este caso, se ajusta el *tokenizador* a las reseñas procesadas, lo que permite crear un índice que asigna un número a cada palabra. Luego, se convierten los textos en secuencias de números basándose en ese índice. Para que todas las secuencias tengan la misma longitud, se recortan o se rellenan con ceros, lo que es importante para que la red *LSTM* pueda procesarlas de manera uniforme.

```python
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Dropout

embedding_dim = 128

model = Sequential()
model.add(Embedding(MAX_NUM_WORDS, embedding_dim, input_length=MAX_SEQUENCE_LENGTH))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Figura 89

Fragmento de código para crear el modelo LSTM

En la Figura 89 se implementa el modelo de *LSTM*. El modelo se construye como una secuencia de capas, donde la primera capa es una capa de *Embedding* que transforma los índices de palabras en vectores numéricos. Aquí, se definen el tamaño del vocabulario y la dimensión de estos vectores, además de establecer la longitud de las secuencias de entrada. A continuación, se incorpora una capa de *LSTM* con 128 unidades, la cual permite al modelo captar la secuencia y el contexto de las palabras, utilizando técnicas de *Dropout* para evitar que el modelo se ajuste demasiado a los datos de entrenamiento. La capa final es una capa densa con una sola neurona y activación *sigmoid*, que convierte la salida en una probabilidad, lo que la hace ideal para problemas de clasificación binaria. Por último, el modelo se configura utilizando el optimizador *Adam* junto con la función de pérdida de entropía cruzada binaria. Como se mencionó en otros modelos el valor de los parámetros responde a la experiencia y un proceso de prueba constante.

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.3, random_state=42)
history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test))
```

Figura 90

Fragmento de código para dividir los datos y entrenar el modelo LSTM

En la Figura 90 se ve la manera de cómo se dividen los datos y se entrena el modelo. Durante el entrenamiento, el modelo procesa los datos en los de treinta y dos ejemplos durante cinco épocas y. además, se incluye un conjunto de validación para evaluar el rendimiento en cada época.

```python
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Accuracy: {accuracy:.4f}")
```

Figura 91

Evaluación del modelo LSTM

En la figura 91 se presenta la evaluación final del modelo. El uso de *LSTM* es adecuado para el procesamiento de lenguaje natural, ya que ayuda a capturar, como se mencionó anteriormente, las relaciones secuenciales y contextuales en el texto. En la Figura 92 se ven los resultados parciales en cada época.

Figura 92

Resultados del modelo LSTM

En la Figura 92 se muestra cómo el modelo mejora a lo largo de cinco épocas de entrenamiento. Inicialmente, tanto en el entrenamiento como en la validación se observan resultados moderados, pero conforme avanza el proceso, el modelo va reduciendo gradualmente la pérdida y aumentando la precisión. Aunque en el conjunto de validación se aprecian algunas fluctuaciones, la tendencia general indica que el modelo está aprendiendo de forma consistente y se acerca a un rendimiento estable.

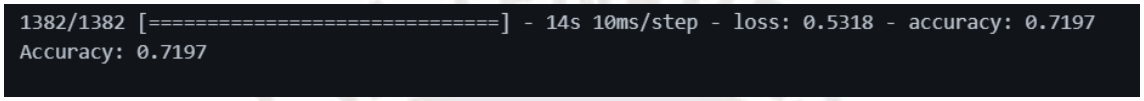


Figura 93

Resultados final del modelo LSTM

En la Figura 93 se muestra cómo se ha entrenado y evaluado el modelo de *LSTM*. Durante el entrenamiento, tanto la perdida como la precisión mejoraron de manera constante. Aunque el rendimiento en el conjunto de validación fue un poco más moderado, se logra una precisión final en el conjunto de prueba cercana al 72% lo que sugiere que el modelo generaliza bien a datos no vistos. Se debe notar que, en este modelo, se hace uso de capa de *embedding* para representar el texto. A diferencia de la técnica *TF-IDF*, que se utiliza en modelos tradicionales, la capa de *embedding* genera representaciones densas y continuas. Estas representaciones no solo reflejan la frecuencia de las palabras, sino que también capturan sus relaciones semánticas y contextuales. Esto es importante en redes neuronales como *LSTM*, ya que permite al modelo entender mejor el significado y el contexto de cada palabra dentro de una secuencia.

5.12 Modelo: Deeplearning – GRU (Gated Recurrent Units)

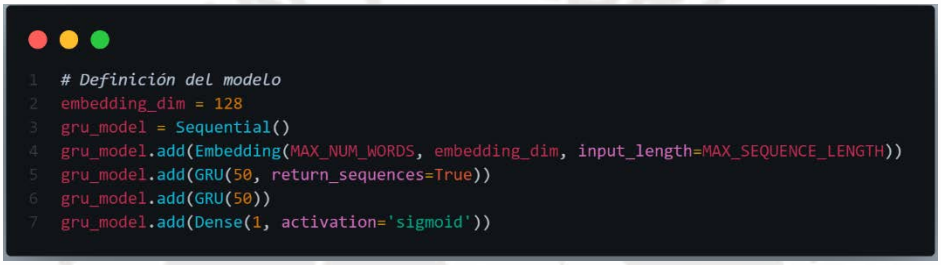
Las *Gated Recurrent Units* (*GRU*) son una variante de las redes neuronales recurrentes que se crearon para solucionar problemas como el desvanecimiento y la explosión del gradiente, introducidas por Cho et al. En 2014. Aunque son similares a las a las *LSTM* en cuanto a que se usan “puertas” para controlar el flujo de información, las *GRU* tienen una arquitectura más simple. Esto significa que combinan la celda de memoria y el estado oculto en una sola unidad, lo que reduce la cantidad de parámetros a ajustar y permite una computación más eficiente y un entrenamiento más sencillo.

```
from keras.models import Sequential
from keras.layers import Embedding, GRU, Dense, Dropout
```

Figura 94

Fragmento de código para importar el modelo GRU

En la Figura 94 se importa el modelo *GRU*. A continuación, como se ve en la Figura 95, la construcción del modelo está utilizando la clase *Sequential* de *Keras*, lo que permite añadir capas de forma ordenada. Primero, se utiliza una capa de *embedding*. Luego se añade una capa de "GRU" con 50 unidades y con el parámetro *return_sequences* igual a *True*, para devolver la secuencia completa de salidas para que la siguiente capa *GRU* pueda procesarla. Posteriormente, se incorpora otra capa *GRU* con 50 unidades que, al no usar *return_sequences*, solo devuelve la última salida de la secuencia, resumiendo la información importante. Por último, la capa de salida es una capa *Dense* con una sola neurona y activación *sigmoid* esto interpreta la salida como una probabilidad entre cero y uno lo que se necesita para este contexto.



```
# Definición del modelo
embedding_dim = 128
gru_model = Sequential()
gru_model.add(Embedding(MAX_NUM_WORDS, embedding_dim, input_length=MAX_SEQUENCE_LENGTH))
gru_model.add(GRU(50, return_sequences=True))
gru_model.add(GRU(50))
gru_model.add(Dense(1, activation='sigmoid'))
```

Figura 95

Fragmento de código para crear el modelo GRU

```
# Compilación del modelo
gru_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Figura 96

Fragmento de código para compilar el modelo GRU

En la Figura 96 se muestra cómo se configura el modelo antes de entrenarlo. Empleando el optimizador "Adam" como se vio en modelos anteriores.

```
# Resumen del modelo
gru_model.summary()
```

Figura 97

Fragmento de código para obtener los resultados del modelo GRU

En la figura 97 se muestra el uso del método *summary*, que proporciona un resumen detallado del modelo. Este resumen incluye el número de parámetros entrenables en cada capa y la forma de las salidas de cada capa, lo cual es permite verificar que la arquitectura del modelo sea la esperada.

```python
# Entrenamiento del modelo
history_gru = gru_model.fit(X_data, y_data, epochs=5, batch_size=32, validation_split=0.3)
```

Figura 98

Fragmento de código para entrenar al modelo GRU

En la Figura 98 se ve el entrenamiento del modelo. Durante el entrenamiento, el modelo procesa los datos en pequeños grupos, lo que facilita la actualización gradual de sus parámetros y un uso eficiente de la memoria.

```python
# Evaluación del modelo
accuracy_gru = gru_model.evaluate(X_test, y_test)[1]
print(f"Accuracy: {accuracy_gru:.4f}")
```

Figura 99

Evaluación del modelo GRU

En la Figura 99 se observa cómo se evalúa el modelo creado. En este punto se evalúa su desempeño en un conjunto de prueba.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 50, 128)           12800

 gru (GRU)                   (None, 50, 50)            27000

 gru_1 (GRU)                 (None, 50)                15300

 dense_4 (Dense)             (None, 1)                 51

=================================================================
Total params: 55151 (215.43 KB)
Trainable params: 55151 (215.43 KB)
Non-trainable params: 0 (0.00 Byte)
```

Figura 100

Resultados del modelo GRU

En la Figura 100 se detalla la arquitectura del modelo de una forma clara el cual consta de una capa *Embedding*, una primera capa de *GRU* y una segunda capa de *GRU* que resume la información, entregando una representación final para que en la capa densa de salida con activación *sigmoid* genere la predicción final.

```
1382/1382 [==============================] - 24s 17ms/step - loss: 0.5852 - accuracy: 0.7117
Accuracy: 0.7117
```

Figura 101

Resultado final del modelo GRU

En la Figura 101 se muestra la evolución del rendimiento del modelo a lo largo de las épocas de entrenamiento. Durante este proceso, el modelo muestra una mejora continua en el conjunto de entrenamiento. Sin embargo, las métricas en el conjunto de validación presentan resultados menos estables, lo que sugiere que el modelo podría estar sobre ajustándose a los datos de entrenamiento y, por ende, podría tener dificultades para generalizar con nuevos datos.

5.13 Comparativa de modelos

En el ámbito del procesamiento de lenguaje natural, es importante seleccionar el modelo adecuado para obtener resultados óptimos. Comparar diversos modelos permite evaluar cómo se desempeñan en un conjunto de datos específico y determinar cuál ofrece mayor precisión y capacidad para manejar datos nuevos. Cada modelo tiene sus propias características y evaluarlos utilizando las métricas, descritas antes, permite identificar claramente cuál se adapta mejor a las particularidades de este conjunto de datos. Es importante destacar que el objetivo es elegir un modelo que no solo maximice la precisión, si no que también sea eficiente y tenga la capacidad de generalizar correctamente con datos no vistos.

```python
# Importando las librerías necesarias
import matplotlib.pyplot as plt

# Datos de precisión para cada modelo
models = ["Logistic Regression", "SVM", "Random Forest", "XGBoost", "XGBoost-OPT", "Neural Network", "GRU"]
accuracies = [0.8012, 0.7972, 0.8022, 0.7813, 0.7787, 0.7921, 0.7153]

# Crear la gráfica nuevamente
plt.figure(figsize=(12, 7))
plt.barh(models, accuracies, color=['blue', 'green', 'red', 'purple', 'purple', 'cyan', 'yellow'])
plt.xlabel('Accuracy')
plt.ylabel('Model')
plt.title('Model Accuracy Comparison')
plt.xlim(0.7, 0.82)  # Limitamos el eje x para mejor visualización
plt.gca().invert_yaxis()  # invertimos el eje y para que el modelo con mejor precisión esté arriba
plt.show()

# Retorna el modelo con la mejor precisión
best_model_index = accuracies.index(max(accuracies))
best_model = models[best_model_index]
best_model
```

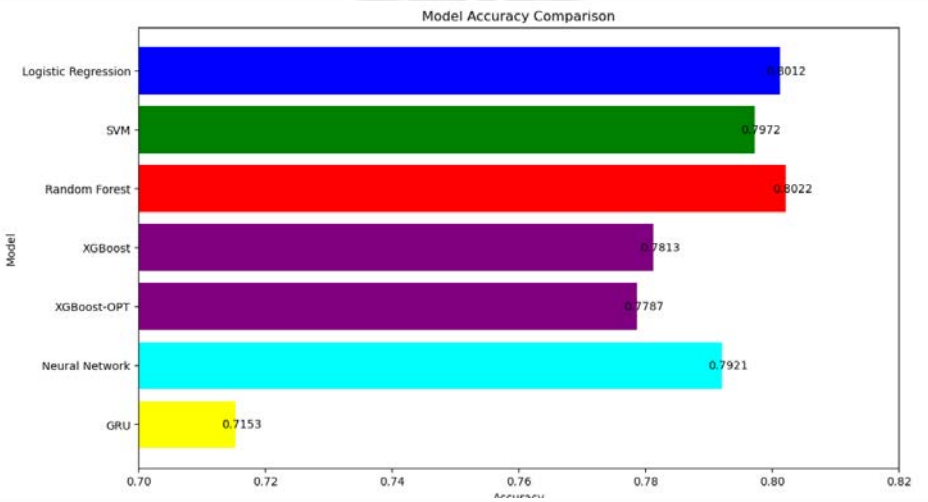
Figura 102

Fragmento de código para comparar la precisión de cada modelo entrenado

En la Figura 102 se muestra el código utilizado para analizar y comparar las precisiones de los distintos modelos evaluados en el presente capítulo. En la gráfica 9 se podrá identificar cuál fue el mejor modelo considerando la precisión.

Gráfica 9

Comparación del resultado de la precisión de cada modelo entrenado



En la Gráfica 9 se muestran los resultados de la comparación entre los distintos modelos. *Random Forest* destaca como el modelo con mayor precisión, seguido de cerca por la Regresión Logística y SVM. Aunque los modelos de *boosting* ("*XGBoost*") y las redes neuronales (tanto la red neuronal como GRU) también presentan buenos rendimientos, ninguno supera a *Random Forest* en este conjunto de datos en particular. Esta comparación resalta la importancia de evaluar y ajustar diferentes modelos y sus hiperparámetros para encontrar el modelo más adecuado para cada tarea. Además, como se ve en la tabla 1, se resume tanto la precisión como el F1-Score de cada modelo, reforzando esta conclusión proporcionando una visión clara que respalda la elección de *Random Forest* como la opción más eficaz en este caso en particular.

Tabla 1

Tabla comparativa final de los modelos de machine learning y deep learning

Modelo	Precisión (Accuracy)	F1-score Ponderado (Weighted)
Logistic Regression	0.8012	~0.80
SVM	0.7972	~0.79–0.80
Random Forest	0.8022	~0.80
XGBoost	0.7813	~0.78
XGBoost-OPT	0.7787	~0.77
Neural Network	0.7921	~0.79
GRU	0.7153	~0.71–0.72

5.14 Guardado del modelo con mejor resultado (Random Forest)

Guardar el modelo con mejor resultado es importante porque permite reproducir los resultados, asegurando que el modelo se pueda aplicar de forma consistente en el futuro. Además, al almacenar el modelo ya entrenado se ahorra tiempo y recursos, ya que no será necesario volver a entrenarlo desde cero. También disponer del modelo guardado facilita su integración en aplicaciones de producción para realizar predicciones en tiempo real si se requiere.

```python
import joblib
import pandas as pd

# Convertir la matriz dispersa a DataFrame
X_test_df = pd.DataFrame(X_test.toarray())

# Guardar el conjunto de test en un archivo CSV
test_data = pd.concat([X_test_df, y_test.reset_index(drop=True)], axis=1)
test_data_path = "test_data.csv"
test_data.to_csv(test_data_path, index=False)

# Guardar el modelo Random Forest
best_model_path = "best_rf_model.pkl"
joblib.dump(rf_model, best_model_path)

test_data_path, best_model_path
```

Figura 103

Fragmento de código para almacenar el modelo de random forest

En la Figura 103 se muestra cómo se guardan tanto los datos de prueba como el modelo *Random Forest* entrenado. Esto permite, como se mencionó anteriormente, evaluar el modelo en un entorno productivo y con nuevos datos además de evitar el reprocesado de los datos.

CAPÍTULO 6 – EVALUACIÓN DEL MODELO RANDOM FOREST

En esta sección se evalúa el rendimiento del modelo *Random Forest*, elegido anteriormente por su desempeño en la clasificación de sentimientos en el conjunto de datos procesado. Esta evaluación es importante para saber cómo se comporta el modelo con datos nuevos, garantizando que no solo funcione bien con los datos de entrenamiento y validación. Para ello, primero se carga el modelo entrenado junto con un conjunto de datos de prueba. Luego se calculan las distintas métricas de evaluación, como la precisión, el *recall*, el F1-Score y el AUC-ROC. Además, se analiza la matriz de confusión para identificar cuántas predicciones fueron correctas o incorrectas en cada clase y detectar patrones de error específicos. Además, se presenta la cura *ROC* y el valor del *AUC*, que muestran la capacidad del modelo para diferenciar entre clases positivas y negativas a distintos umbrales de decisión. Este enfoque no solo mide el rendimiento del modelo, sino que también ayuda a tomar decisiones para optimizarlo, asegurando que sea robusto, preciso y adecuado para su implementación.

6.1 Carga del modelo y conjunto de datos

```python
import joblib
import pandas as pd

# Cargar el modelo
rf_model = joblib.load('best_rf_model.pkl')

# Cargar el conjunto de datos de prueba
test_data = pd.read_csv('test_data.csv')

X_test = test_data.drop(columns=['label'])
y_test = test_data['label']
```

Figura 104

Fragmento de código para cargar el modelo de random forest

En la figura 104 se describe el proceso de carga y preparación del modelo *Random Forest* previamente guardado para su evaluación en un conjunto de datos de prueba. Este proceso asegura que tanto el modelo como los datos de prueba estén en el formato correcto y listos para calcular las métricas de rendimiento.

```python
y_pred = rf_model.predict(X_test)
```

Figura 105

Fragmento de código para obtener las predicciones del modelo de random forest

En la Figura 105 se observa la instrucción que genera las predicciones sobre el conjunto de datos de prueba, permitiendo de esta manera evaluar cómo de bien generaliza el modelo *Random Forest* a nuevos datos.

6.2 Cálculo de métricas de evaluación

```python
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Calcular la precisión
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Reporte de clasificación
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)

# Matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Figura 106

Fragmento de código para evaluar las predicciones del modelo de random forest

En la Figura 106 se muestra el proceso para realizar una evaluación detallada del modelo "Random Forest" en el conjunto de prueba utilizando varias métricas para comprender su rendimiento. En primer lugar, se calcula la precisión. Luego, se genera un informe de clasificación que ofrece datos más específicos para cada clase. Además, se genera la matriz de confusión el cual muestra cuántas predicciones fueron correctas o incorrectas para cada clase.

```
Accuracy: 0.8022
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.69      0.74     17654
           1       0.81      0.88      0.84     26557

    accuracy                           0.80     44211
   macro avg       0.80      0.78      0.79     44211
weighted avg       0.80      0.80      0.80     44211

Confusion Matrix:
[[12198  5456]
 [ 3289 23268]]
```

Figura 107

Resultados de las métricas obtenidas por el modelo de random forest

En la Figura 107 se muestra un análisis detallado de los resultados del modelo *Random Forest*. En términos generales, el modelo logra clasificar correctamente el 80.22% de las instancias del conjunto de prueba, lo que da una idea del buen rendimiento que proporciona. Sin embargo, esta métrica general no dice cómo se comporta el modelo en cada clase por separado. Por ejemplo, para la clase positiva, el modelo presenta un F1-Score de 0.84 y un *recall* del 0.88, lo que significa que identifica la mayoría de las instancias positivas, un aspecto importante en la clasificación de sentimientos. En cambio, para la clase negativa, aunque la precisión esta alrededor de 0.79, el *recall* es más bajo, alrededor de 0.69, lo que indica que el modelo falla en capturar algunas instancias negativas.

6.3 Matriz de confusión

La matriz de confusión se emplea para evaluar qué tan bien está funcionando un modelo de clasificación. Muestra en una tabla las predicciones realizadas por el modelo y las compara con las etiquetas reales del conjunto de datos lo que permite ver cuántas son correctas o incorrectas. Además, con esta matriz se evalúa si el modelo está confundiendo sentimientos positivos con negativos o viceversa.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, roc_curve, auc

# 1. Matriz de Confusión
y_pred = rf_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(7, 5))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Matriz de Confusión')
plt.colorbar()
classes = [0, 1]
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)

# Loop sobre las dimensiones de la matriz de confusión para agregar los números en cada celda
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'), horizontalalignment="center", color="white" if cm[i, j] > cm.max() / 2 else "black")

plt.tight_layout()
plt.ylabel('Etiqueta verdadera')
plt.xlabel('Etiqueta predicha')
plt.show()
```

Figura 108

Fragmento de código para generar la matriz de confusión del modelo de random forest

En la Figura 108 se muestra cómo se genera y visualiza la matriz de confusión para el modelo *Random Forest* al comparar las etiquetas reales con las predicciones. Con esto se crea una gráfica que colorea cada celda según la cantidad de aciertos o errores, de manera que se etiqueta las clases en los ejes y se coloca los valores correspondientes en cada posición.

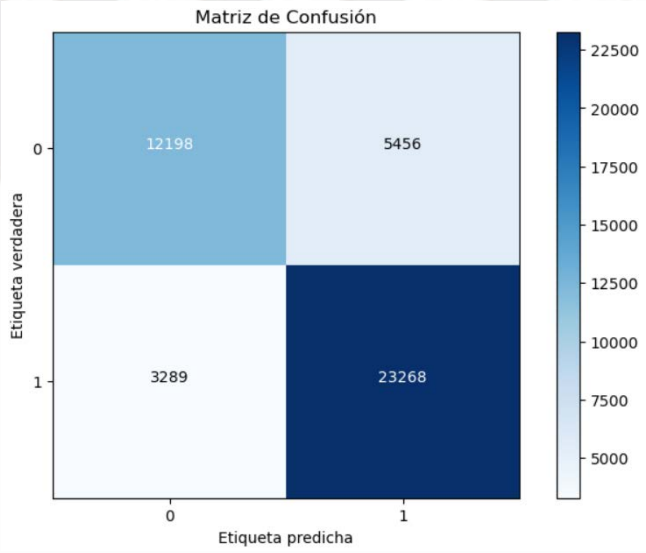


Figura 109

Matriz de confusión del modelo de random forest

En la Figura 109 se muestra la matriz de confusión generada en el paso anterior. En esta matriz, el eje vertical muestra las etiquetas reales (0 para negativos y 1 para positivos), y el eje horizontal muestra las etiquetas que el modelo predice. Se observa, que el modelo clasificó correctamente un gran número de casos negativos y positivos, con 12198 verdaderos negativos y 23268 verdades positivos. Sin embargo, también se detectan errores: hay 5456 casos en los que se predijo incorrectamente un sentimiento negativo cuando era positivo, y 3289 casos en los que se clasificó erróneamente como positivo cuando en realidad era negativo. Sin embargo, debida a la gran cantidad de datos procesados y los resultados obtenidos este modelo responde de manera adecuada y se encuentra dentro del rango permitido para ser implementado en producción.

6.4 Curva ROC y AUC

En el contexto del procedimiento de lenguaje natural, esta métrica resulta útil ya que los datos textuales suelen estar desbalanceados; por ejemplo, puede haber muchos más sentimientos positivos que negativos. Esto hace que métricas como la precisión puedan no ser tan exactas. La curva *ROC* y el *AUC* ofrece una evaluación más completa al mostrar la capacidad del modelo para diferenciar entre clases de distintos umbrales.

```python
# 2. Curva ROC y AUC
y_pred_prob = rf_model.predict_proba(X_test)[:, 1]  # Probabilidades para la clase positiva
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 7))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='Curva ROC (área = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC')
plt.legend(loc="lower right")
plt.show()
```

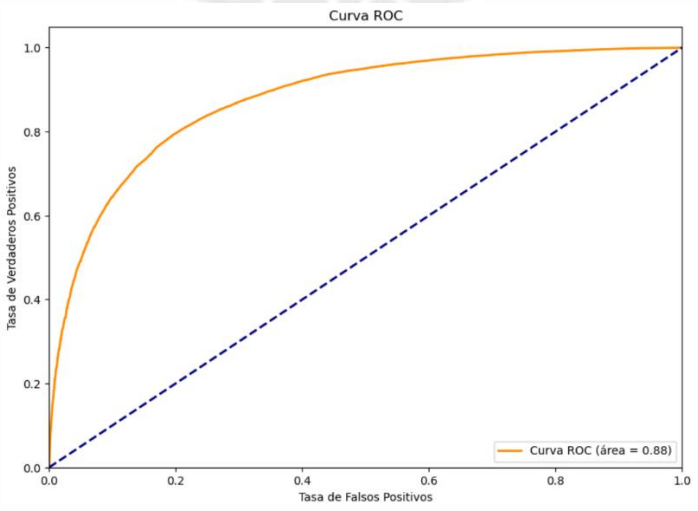
Figura 110

Muestra del código para visualizar las curvas ROC y AUC del modelo de random forest

En la Figura 110 se muestra cómo se genera las curcas *AOC* y *ROC* para el modelo *Random Forest*. Así mismo, el *ROC* ayuda a visualizar cuán bien el modelo distingue entre las dos clases y, por otro lado, el *AUC* resume esta curva en un solo número, donde un valor de uno representa un modelo perfecto y un valor de 0.5 indica que el modelo no es mejor que adivinar al azar.

Gráfica 10

Curva ROC y AOC del modelo de random forest



En la gráfica 10 se observa la curva *ROC* (*Receiver Operating Characteristic*) y el Área bajo la Curva (*AUC*). La curva *ROC* ilustra cómo varía la tasa de verdaderos positivos en comparación con la tasa de falsos positivos a diferentes umbrales de decisión, mientras que la línea diagonal representa el desempeño de un clasificador aleatorio. El valor de "AUC" es de 0.88 lo que indica que el modelo es muy bueno para diferenciar entre sentimientos positivos y negativos. En otras palabras, el modelo, según estos resultados, identifica los eficazmente los casos positivos (alto *recall*) y, al mismo tiempo, comete pocos errores al etiquetar incorrectamente los negativos (bajo FPR). Aunque el modelo *Random Forest*, en este caso, demuestra ser eficaz para el análisis de sentimiento, existe margen para mejorar su rendimiento y aplicabilidad mediante técnicas avanzadas y un monitoreo continuo. No obstante, los resultados actuales indican que el modelo está listo para desplegarse en un entorno real, donde pueda procesar nuevas reseñas de productos. Esto va a permitir a la empresa anticipar compañas de marketing y, en consecuencia, mejorar la experiencia de sus consumidores. Es igualmente relevante tener en cuenta el conocimiento del negocio y del sector específico al analizar los resultados. Esto va más allá de la parte técnica, ya que conocer el sector del luego, la marca, los objetivos de la empresa asegura una interpretación precisa de los resultados del modelo.

CAPÍTULO 7 – ANÁLISIS DE COSTOS

En este capítulo se realiza un análisis de los costos relacionados con la implementación de un proyecto de NLP, centrado en el análisis de sentimiento. Se detallen los requisitos necesarios para desplegar modelos de *machine learning*, incluyendo las tecnologías y herramientas requeridas, así como la infraestructura necesaria para poner en funcionamiento estos modelos de manera escalable. Se abordan componentes como la selección de plataformas para desarrollo y despliegue, la necesidad de recursos informáticos adecuados, el almacenamiento y gestión del dato, y consideraciones de seguridad y mantenimiento. Además, se exploran las opciones entre servicios en la nube e infraestructura local, presentando estimaciones de costos para cada alternativa, considerando factores como el procesamiento de grandes volúmenes de datos, la capacidad de escalabilidad del servicio y la integración con sistemas existentes.

El análisis cubre varios aspectos, desde la fase inicial de desarrollo hasta el despliegue y el mantenimiento continuo. Esto incluye la evaluación de recursos humanos necesarios, como científicos de datos, ingenieros de *machine learning* y desarrolladores de software. Además, se considera los costos relacionados con el entrenamiento de modelos, que pueden incluir el uso de infraestructura computacional avanzada como "*GPUs*" y "*TPUs*", así como la adquisición de grandes conjuntos de datos etiquetados para entrenar y validar los modelos.

7.1 Recursos humanos

Para implementar un proyecto de análisis de sentimiento es importante contar con un equipo multidisciplinario. Esto incluye científico de datos, ingenieros de datos e ingenieros de *machine learning*, los cuales se encargan de investigar y ajustar los modelos. Se puede destacar que los científicos de datos y los ingenieros de *machine learning* se dedican a elegir los algoritmos adecuados, ajustar sus parámetros y validar los resultados, lo que les permite mejorar constantemente la precisión y eficiencia del modelo. Además, es importante que tengan experiencia en, específicamente, la implementación de algoritmos de procesamiento de lenguaje natural (NLP). Aunque también deben intervenir más disciplinas éstos son los más solicitados al momento de implementar un proyecto de esta naturaleza y, debido a que no se cuenta con personal calificado, muchas veces son los mejores pagados en el sector.

7.2 Infraestructura computacional

La infraestructura juega un rol importante para poner en marcha los modelos desarrollados, especialmente en la industria del lujo, donde la rapidez y la precisión son factores importantes. Contar con un entorno tecnológico adecuado garantiza que el entrenamiento y la realización de predicciones sean eficientes y escalables, incluso cuando se manejan grandes volúmenes de datos. Por otra parte, el hardware debe disponer de servidores equipados con "*GPU*" y "*TPU*". Las "*GPU*" facilitan el procesamiento en paralelo, lo que permite realizar múltiples cálculos al mismo tiempo lo que reduce el tiempo de entrenamiento del modelo y, por otro lado, las "*TPU*", diseñadas específicamente para tareas de *deep learning*, ofrecen un rendimiento aún mayor en este tipo de operaciones. El almacenamiento también es importante, ya que se necesita un sistema robusto para gestionar una gran cantidad de datos textuales y guardar los modelos entrenados junto con sus diferentes versiones. Es importante contar con solucione que sean escalables y eficientes, como bases de datos *NoSQL* o incluso considerar el uso de la nube lo que puede permitir que el costo operativo varíe en función de los recursos utilizados.

7.3 Software y herramientas

Para desarrollar, entrenar, desplegar y gestionar modelos de análisis de sentimientos basados en NLP, se debe contar con las herramientas y software adecuados para el mismo. Por un lado, las herramientas de código abierto, como *TensorFlow* y *PyTorch*, ayudan a construir y ajustar los modelos. Otras herramientas como *NLTK* y *SpaCy* se emplean para el preprocesamiento y análisis de texto, las cuales ofrecen funciones de *tokenización*, *lematización* y análisis sintáctico.

Además, en el despliegue de los modelos se utilizan plataformas que aseguran la capacidad de crecimiento y el rendimiento del sistema. Tecnologías como *Kubernetes* y *Docker* permiten organizar y encapsular aplicaciones en contenedores, asegurando que los modelos se implementen de manera uniforme en cualquier entorno. Servicios en la nube como *Amazon Web Services*, *Google Cloud Platform* y *Microsoft Azure* proporcionan infraestructura que ofrecen recursos computacionales a demanda.

7.4 Requerimiento para el despliegue de modelos

El despliegue efectivo de estos modelos requiere de una planificación que cubra el entrenamiento, la implementación y la operación fluida. Utilizando contenedores como *Docker* junto con herramientas de orquestación como *Kubernetes*, se aseguran de que los modelos sean portátiles y consistentes en diversos entornos, facilitando su administración y escalabilidad automática. Además, el uso de instancias de computación en la nube y servicios gestionados de *machine learning* ofrece flexibilidad para ajustar recursos según demanda y optimizar el rendimiento del modelo. El término de gestión, plataformas en la nube y bases de datos *NoSQL* como *Amazon S3*, *Google Cloud Storage* y *MongoDB* permiten almacenar de manera segura tanto datos textuales como resultados de las predicciones del modelo. También son fundamentales herramientas de *ETL* (*Extract transform and load*) para garantizar que los datos estén preparados para el entrenamiento de los modelos. Por otra parte, para asegurar la seguridad y cumplir con regulaciones de privacidad como *GDPR* (*General Data Protection Regulation*) y el *CCPA* (*California Consumer Privacy Act*), se implementan políticas estrictas de control de acceso y autenticación Este enfoque integral asegura la protección y el cumplimiento normativo de los modelos.

7.5 Tecnologías recomendadas

Para seleccionar la tecnología adecuada para implementar un proyecto de análisis de sentimiento (NLP). Se recomienda usar herramientas de *machine learning* como *TensorFlow* y *Pytorch*, junto con modelos preentrenados de *Hugging Face Transformes*, estos últimos no se emplearon en la presente tesis, pero en un entorno productivo y con mayores recursos es una buena opción. Además, contar con plataformas en la nube como *AWS*, *GCP* o *Azure* hace que el modelo pueda ser escalable y se pueda reproducir a demanda para evitar costos innecesarios. Además, se debe considerar herramientas de integración continua como *Jenkins* o *Gitlab* para automatizar el proceso. Además, al ser un proyecto multidisciplinario cada una de estas herramientas debe tener un especialista que sea capaz de implementar de manera adecuada la infraestructura y recomendar las tecnologías a emplear.

7.6 Estimación de costos para el desarrollo de un proyecto de NLP

Los costos asociados varían dependiendo de factores como el tamaño del equipo, las tecnologías utilizadas y la infraestructura necesaria para entrenar y desplegar los modelos. A continuación, se presenta una estimación de los costos promedio para llevar a cabo un proyecto de este tipo durante un período de 12 meses. Estos costos están divididos en recursos humanos, tecnologías y otros elementos clave, y se detallan en la tabla 2.

Tabla 2

Costos asociados, en dólares, anuales para implementa un proyecto de procesamiento de lenguaje natural (NLP)

Recursos / Tecnología	Descripción	Costo Promedio Mensual (USD)	Costo Total Proyecto (USD)
RECURSOS HUMANOS			
Científico de Datos	Investigación y desarrollo de modelos, selección de algoritmos	10,000	120,000
Ingeniero de Machine Learning	Ajuste de hiperparámetros, optimización de modelos	10,000	120,000
Desarrollador de Software	Integración de modelos en aplicaciones y sistemas	8,000	96,000
Especialista en Infraestructura	Configuración y Mantenimiento de la infraestructura	9,000	108,000
TECNOLOGÍAS Y HERRAMIENTAS			
Plataformas en la Nube (AWS, GCP, AZURE)	Servicios en la nube para almacenamiento y computación	5,000	60,000
Herramientas de Desarrollo (IDE, Git)	Software de desarrollo y control de versiones	500	6,000
Licencias de Software (NLP Libraries)	Licencias para bibliotecas y herramientas de NLP	1,000	12,000
INFRAESTRUCTURA Y HARDWARE			
Servidores de Entrenamiento	Servidores GPU para entrenamiento de modelos	4,000	48,000
Servidores de Producción	Servidores para desplegar modelos en producción	3,000	36,000
Almacenamiento de Datos	Bases de datos y almacenamiento de grandes volúmenes de datos	2,000	24,000
OTROS COSTOS			
Consultoría y Soporte	Servicios de consultoría externa y soporte técnico	2,000	24,000
Formación y Capacitación	Capacitación continua para el equipo en nuevas tecnologías y técnicas	1,000	12,000
COSTOS TOTALES			726,000

La tabla proporciona una estimación integral completa de los costos relacionados con el desarrollo de un proyecto de NLP para el análisis de sentimientos. La inversión en personal altamente capacitado, tecnologías avanzadas y una infraestructura robusta asegura, en gran medida, el éxito del proyecto. Este análisis brinda a los interesados una comprensión clara de las necesidades financieras y tecnológicas, facilitando una planificación y ejecución más efectiva del proyecto. Además, se observa que los costos asociados a proyectos de esta índole son significativos, lo cual posiciona a las empresas con mayores recursos financieros en la vanguardia de estas tecnologías y, por ende, tener una ventaja competitiva en el sector.

7.7 Resumen del análisis de costos para un proyecto de análisis de sentimiento empleando NLP

Contar con científicos de datos especializados en el sector, ingenieros de *machine learning* y desarrolladores permite desarrollar modelos precisos e integrarlos en aplicaciones prácticas. Además, utilizar expertos en infraestructura y plataformas en la nube asegura la capacidad para manejar grandes volúmenes de datos y cargas de trabajo intensivas, mejorando así el rendimiento y reduciendo costos. Esta inversión no solo garantiza la creación de modelos confiables, si no que también respalda la toma de decisiones informadas en áreas de marketing, atención al cliente y desarrollo de productos, fortaleciendo la competitividad a largo plazo para la empresa.

CONCLUSIONES Y RECOMENDACIONES

En esta sección se presentan conclusiones derivadas del desarrollo y evaluación del modelo de *machine learning* utilizado para el análisis de sentimientos. Se detallan los principales hallazgos alcanzados a lo largo de esta tesis, así como el impacto de los resultados obtenidos en nuestra comprensión de las percepciones de los consumidores en el sector del lujo. Además, se da recomendaciones detectadas, a lo largo del desarrollo, con el objetivo de optimizar las estrategias de las marcas y mejorar la experiencia de los consumidores.

8.1 Conclusiones

- Esta tesis ha desarrollado un sistema avanzado de análisis de sentimientos que utiliza técnicas de procesamiento de lenguaje natural y aprendizaje automático para clasificar reseñas que realizan los consumidores a los productos adquiridos en el sector del lujo.
- Se logró profundizar en la comprensión de las emociones del consumidor, permitiendo ajustar las interpretaciones de los modelos con la finalidad de otorgar a la empresa una herramienta que le permita ajustar estrategias de producto y comunicación para superar las expectativas de sus consumidores.
- El análisis exploratorio de datos reveló patrones y sesgos clave, fundamentales para desarrollar un modelo que distingue entre sentimientos positivos y negativos. Y al considerar que el sector en el cual se desarrolla esta tesis, sector del lujo, pone en evidencia que mayormente los consumidores que adquieren productos de alta gama tienen una mayor satisfacción estadísticamente debido a la experiencia que ofrecen estas marcas más allá de los productos.
- Tras comparar diversos algoritmos, incluyendo modelos tradicionales y redes neuronales, se identificó al modelo *Random Forest* como el más robusto y preciso para este conjunto de datos en específico.
- Se exploraron múltiples modelos de *machine learning* y se destacaron los más efectivos para cada tipo, enfatizando la importancia de la experimentación continua y la optimización de los hiperparámetros.
- El sistema desarrollado no solo ofrece una herramienta precisa para el análisis de sentimientos, sino que también proporciona una base sólida para decisiones informadas en un mercado competitivo y dinámico.
- Se puso en evidencia que la metodología de *Gridsearch* para encontrar los mejores parámetros es un proceso que requiere tiempo, pero que, si se realiza de manera adecuada y con los recursos necesarios, puede beneficiarse significativamente a los modelos desarrollados.
- Las conclusiones derivadas permiten formular recomendaciones prácticas para optimizar estrategias de marketing y mejorar la experiencia de los consumidores en el sector del lujo.

8.2 Recomendaciones

A pesar de los buenos resultados obtenidos en el desarrollo de la presente tesis, se identificaron áreas de mejora para el sistema de análisis de sentimiento:

- Explorar técnicas más avanzadas de NLP, como el uso de *embeddings* preentrenados o arquitecturas más sofisticadas (por ejemplo, RNNs o transformadores), para capturar mejorar el contexto y las sutilezas del lenguaje.
- Implementar un sistema de monitoreo continuo para revisar el rendimiento del modelo en tiempo real y hacer ajustes según sea necesario, asegurando su relevancia y actualización constante frente a cambios en los datos.
- Analizar activamente el *feedback* de los consumidores para identificar áreas de mejora que no se evidencien únicamente con análisis técnicos.
- Adoptar un sistema de aprendizaje activo, donde el modelo solicite etiquetas para ejemplos inciertos, con el fin de mejorar progresivamente su precisión.

- Considerar la incorporación de modelos de lenguaje grande (*LLM*) como *GPT*, *LLAMA* o *BERT*, conocidos por su capacidad para comprender y procesar lenguaje para potenciar aún más el rendimiento del sistema.

Estas recomendaciones podrían fortalecer significativamente la precisión y eficiencia del análisis de sentimientos, permitiendo a la marca optimizar sus estrategias de manera aún más efectiva.

BIBLIOGRAFÍA

1. Chomsky, N. (1957). *Syntactic structures*. Mouton.
2. Fodor, J. A. (1975). The language of thought. Crowell.
3. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735
4. Koehn, P. (2009). Statistical machine translation. Cambridge University Press.
5. Mikolov, T., Yih, W. T., & Zweig, G. (2013). Linguistic regularities in continuous space word representations. Proceedings of the NAACL-HLT 2013, 746–751. https://doi.org/10.3115/3046249.3046497
6. Minsky, M. (1961). Steps toward artificial intelligence. Proceedings of the Institute of Radio Engineers, 49(1), 8-30. https://doi.org/10.1109/JRPROC.1961.287775
7. Shortliffe, E. H. (1980). Mycin: A knowledge-based expert system. Elsevier.
8. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS 2017), 6000–6010. https://doi.org/10.5555/3295222.3295342
9. Weizenbaum, J. (1966). ELIZA—A computer program for the study of natural language communication between man and machine. Communications of the ACM, 9(1), 36-45. https://doi.org/10.1145/365230.365257
10. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shinn, A., Shazeer, N., & Sutskever, I. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165. https://arxiv.org/abs/2005.14165
11. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2.a ed.). O'Reilly Media.
12. Müller, A. C., & Guido, S. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media.
13. Scikit-learn Developers. (n.d.). *Model evaluation: Quantitative measures*. [Documentación de scikit-learn]. Recuperado de https://scikit-learn.org/stable/modules/model_evaluation.html
14. Scikit-learn Developers. (n.d.). *Precision, recall, F1-score*. [Documentación de scikit-learn]. Recuperado de https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html
15. **Bird, S., Loper, E., & Klein, E. (2009).** *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media.
16. Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*. Addison-Wesley.
17. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient estimation of word representations in vector space*. ArXiv Preprint ArXiv:1301.3781.
18. Pennington, J., Socher, R., & Manning, C. D. (2014). *Glove: Global vectors for word representation*. En *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).
19. Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). *Enriching Word Vectors with Subword Information*. Transactions of the Association for Computational Linguistics, 5, 135–146.
20. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.
21. Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural Computation, 9(8), 1735–178