**PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR**

**FACULTAD DE INGENIERÍA**

**ESCUELA DE SISTEMAS**

**TEMA:**

# Análisis de sentimientos de percepción sobre el socialismo en Latinoamérica en la última década

**AUTOR:**

PUPIALES QUINATOA JAVIER ALEXANDER

**QUITO DM, julio 2025**

## DEDICATORIA

Dedico este trabajo a Dios, por darme la vida, salud y la fortaleza para continuar este camino con propósito. A mis padres, por brindarme la educación, el amor y los recursos necesarios para formarme, así como por estar pendientes de mí cada día con entrega incondicional. A mis hermanos, por acompañarme, alentarme y confiar en mí a lo largo de esta etapa. Y a todas las personas que me ofrecieron su apoyo sincero —compañeros, amigos y quienes, con palabras o acciones, me animaron a seguir adelante

# AGRADECIMIENTO

Agradezco en primero a Dios, por haberme dado vida, salud y las capacidades necesarias para culminar esta etapa académica. Su guía ha sido mi fuerza constante. A mis padres, quienes con su amor, sacrificio y compromiso hicieron posible esto. Gracias por estar conmigo en cada paso, por su ejemplo y por enseñarme que el trabajo duro trae consigo grandes recompensas. A mis profesores, por compartir su conocimiento con pasión y dedicación. Su enseñanza despertó en mí un profundo interés por el mundo de la programación, los servicios web, las bases de datos, la ciencia de datos y la inteligencia artificial. Su influencia fue importante para que encontrara mi vocación profesional. También quiero agradecer a mis compañeros de carrera y a todas las personas que me ofrecieron su apoyo, sus consejos o su compañía. Cada gesto fue un gran motivo que me impulsó a seguir.

A todos ustedes, gracias.

# RESUMEN

El presente proyecto tiene como objetivo analizar las percepciones ciudadanas sobre el socialismo en Latinoamérica en los últimos años, mediante técnicas de clasificación de análisis de sentimientos y minería de datos aplicadas a comentarios extraídos de redes sociales. Para un mejor control del proyecto se usó una parte de la metodología CRISP-DM hasta la fase 5 de evaluación, empezando desde la comprensión del problema hasta la construcción y comparación de modelos de aprendizaje automático. El corpus de este proyecto empezó en el recolectado de datos, utilizando comentarios sobre líderes políticos, discursos y documentales relacionados con el Socialismo del Siglo XXI. Se implementaron múltiples algoritmos tradicionales de clasificación de texto y modelos de lenguaje a gran escala (LLMs). Este estudio saca a luz, el valor del análisis automatizado de opiniones políticas y muestra como los modelos de IA, pueden superar a los enfoques tradicionales en tareas de clasificación de sentimientos en contextos sociales.

## ABSTRACT

The present project aims to analyze citizen perceptions about socialism in Latin America in recent years, through classification techniques of feelings analysis and data mining applied to comments extracted from -dedes. For better project control, a part of the CRISP-DM methodology was used to phase 5 evaluation, starting from the understanding of the problem to the construction construction and comparison of authetic learning models. The corpus of this project began in the data collected, using comments on political leaders, speeches and documenting them related to the socialism of the 21st century. Multiple traditional algorithms of text -class classification and large -scale language models (LLMS) are implemented. This study brought to light, the value of the automated political and muiestra opinion analysis as AI models, can overcome crodition approaches in classification of feelings in social contexts.

## Índice

**Tabla de figuras**

**Tabla de tablas**

# TEMA: ANÁLISIS DE SENTIMIENTOS DE PERCEPCIÓN SOBRE EL SOCIALISMO EN LATINOAMÉRICA EN LA ÚLTIMA DÉCADA

# CAPÍTULO I: INTRODUCCIÓN

## Marco de referencia

- Justificación

El movimiento del Socialismo del Siglo XXI ha generado gran controversia en la opinión pública del pueblo Latino reflejada por medio de redes sociales. Este estudio no solo permite conocer cómo los ciudadanos piensan sobre esta corriente, sino que también da una oportunidad para aplicar ciencia de datos en un ámbito político con un gran impacto regional. Además, hay oportunidad para comparar metodologías de clasificación emocional desde enfoques tradicionales hasta modelos basados en IA, contribuyendo al desarrollo de soluciones técnicas en el ámbito del análisis automatizado de texto. Su aporte es tanto académico como social, ya que permite entender de forma estructurada lo que varias personas piensan sobre el socialismo en un espacio digital no mediado.

## Planteamiento del problema

En las últimas décadas, Latinoamérica ha vivido un proceso político, económico y social debido a la presencia de gobiernos que forman parte del autodenominado grupo del "Socialismo del Siglo XXI". Importantes países como Venezuela, Bolivia, Ecuador y en menor medida, Argentina o Colombia, adoptaron una postura en la que promueven una mayor intervención del Estado, programas de redistribución social, nacionalización de sectores estratégicos y reformas constitucionales bajo un enfoque de "justicia social". Este tipo de

gobiernos generaron gran controversia y tuvieron gran apoyo por parte de sectores populares, pero también duras críticas ya que los percibieron como regímenes autoritarios, corruptos o ineficaces para llevar adelante un país económicamente.

Durante esta etapa, se produjo una gran polarización política y la fragmentación ideológica se intensificaron por país y por región, dejando dos bandos democráticos en cada región, que pueden identificarse como los "corrientes" de derecha, socialistas o con tendencia de izquierda alrededor de la cual, según se evidencia en distintas plataformas de redes sociales, se pueden encontrar manifestaciones de apoyo y rechazo a esta tendencia. Los medios de comunicación tradicionales reflejaron esta brecha política, pero con el avance de la tecnología, las opiniones y expresiones se han hecho más visibles gracias al acceso a Internet y las redes sociales, plataformas como Facebook, TikTok, X y YouTube se convirtieron en un espacio clave donde millones de usuarios expresaran abiertamente sus percepciones, emociones y valoraciones frente a los gobiernos actuales en Latinoamérica. Sin embargo, estas manifestaciones no han sido objeto de un análisis profundo y sistemático mediante el uso de herramientas digitales y científicas.

Actualmente, hay pocos estudios que exploren a detalle, desde una perspectiva cuantitativa, el sentimiento popular a cerca del socialismo en América Latina. La mayoría de los análisis que existen solo provienen de encuestas tradicionales, investigaciones de opinión pública estructurada o interpretaciones mediáticas, los cuales, si bien si dan un buen aporte en ciertos casos, pero sus estudios resultan ser escasos al momento de capturar la diversidad emocional, la opinión en el discurso ciudadano y la evolución del pensamiento colectivo a lo largo del tiempo. En contraste, las plataformas digitales como Facebook, YouTube, TikTok hoy en día son una fuente crucial para expresar su punto de vista sobre temas políticos.

El gran volumen de comentarios disponibles en videos sobre líderes del Socialismo del Siglo XXI, documentales, noticieros, entrevistas o reacciones de la gente, constituye una investigación valiosa que aún no ha sido sistemáticamente analizado a profundidad. Este vacío deja en duda preguntas fundamentales: ¿Qué emociones predominan (positivas, negativas, neutras) cuando los ciudadanos opinan sobre el socialismo en distintos contextos? ¿Cómo se diferencia la ideología de la gente cuando se habla de lideres políticos que fomentan este régimen? ¿Existen cambios significativos entre países como Venezuela, Ecuador o Bolivia? ¿Ha cambiado el sentimiento colectivo del socialismo tras eventos como elecciones, crisis económicas, protestas sociales o migraciones?

De este análisis se desprenden los siguientes problemas principal y secundarios:

**Problema principal:**

¿Cuál ha sido la percepción de los ciudadanos latinoamericanos sobre el socialismo durante la última década?

**Problema secundario 1:**

¿Qué emociones predominan (positivas, negativas, neutras) en los comentarios sobre el socialismo latinoamericano?

**Problema secundario 2:**

¿Cómo se puede construir un modelo de aprendizaje automático que permita clasificar comentarios relacionados con el socialismo en Latinoamérica en la última década?

**Problema secundario 3:**

¿Cuál es el desempeño de modelos de análisis de sentimientos, incluidos los modelos de lenguaje a gran escala (LLMs), al clasificar comentarios sobre el socialismo en Latinoamérica?

**Problema secundario 4:**

¿Qué diferencias de rendimiento se evidencian entre los modelos clásicos de análisis de sentimiento y un modelo basado en inteligencia artificial (LLM) al ser aplicados al mismo conjunto de comentarios?

**Objetivos**

- **Objetivo general**

  Analizar las percepciones ciudadanas sobre el socialismo en Latinoamérica en los últimos diez años mediante técnicas de minería de texto y análisis de sentimiento aplicados a comentarios extraídos de redes sociales.

- **Objetivos específicos**

  - ***Objetivo específico 1:***

    Extraer comentarios relacionados al socialismo en Latinoamérica en redes sociales.

  - ***Objetivo específico 2:***

    Diseñar y entrenar un modelo de aprendizaje automático que permita clasificar comentarios sobre el socialismo según su polaridad emocional.

- ***Objetivo específico 3:***

Evaluar el rendimiento de distintos modelos de análisis de sentimientos, incluidos modelos de lenguaje a gran escala (LLMs), en la clasificación de comentarios relacionados con el socialismo en Latinoamérica.

- ***Objetivo específico 4:***

Comparar el rendimiento de modelos clásicos de análisis de sentimiento con un modelo basado en IA (LLM), utilizando métricas de evaluación sobre un mismo conjunto de datos.

## Antecedentes

Diversos estudios han abordado el impacto del Socialismo del Siglo XXI en América Latina desde una perspectiva política o económica. Sin embargo, pocos han examinado la percepción pública expresada en entornos digitales utilizando herramientas automatizadas. Investigaciones como la de Liu (2012) sobre análisis de sentimientos, y modelos como BERT, han demostrado su utilidad en múltiples áreas, pero su aplicación en temas ideológicos sigue siendo un campo poco explorado. Asimismo, estudios comparativos entre modelos tradicionales y modelos de lenguaje a gran escala (LLMs) son incipientes, especialmente en contextos políticos latinoamericanos.

## Alcance

Este proyecto se centró en el análisis de percepciones ciudadanas sobre el socialismo en Latinoamérica mediante comentarios en redes sociales, principalmente YouTube. Se recopilaron datos públicos relacionados con figuras políticas representativas de esta ideología,

y se procesaron mediante técnicas de minería de texto. Se entreno seis modelos de análisis de sentimientos: 4 modelos clásicos y 2 modelos utilizando una IA basada en lenguaje (LLM) como ChatGPT o DeepSeek, integrado mediante Jupyter Notebooks. El objetivo es comparar el desempeño de estos modelos en tareas de clasificación de emociones (positiva, negativa, neutra). El marco metodológico utilizado toma como referencia la metodología de CRISP-DM, no completamente, ya que no se toma en cuenta la fase 6 "Despliegue". No se desarrollará ninguna aplicación web ni se recopilarán datos privados o confidenciales.

# CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA

## Marco teórico

### Socialismo del Siglo XXI

El origen del socialismo nace a partir del siglo XIX entre grandes abusos del capitalismo industrial, el cual tiene raíces filosóficas y económicas. El concepto inicial sobre el socialismo viene de la antigua Grecia y donde se destacan pensadores utópicos Thomas More, Saint-Simon, Fourier y Owen (History.com Editors, 2018). El socialismo moderno empieza a evolucionar en la época de los 80`s en Europa, Francia, con autores como Robert Owen, Pierre Leroux y Marie Reybaud con el fin de protesta contra las condiciones obreras a causa de la revolución industrial.

El socialismo del siglo XXI fue propuesto por Heinz Dieterich Steffan en el año de 1996, la cual fue evolucionada e implementada por el expresidente de Venezuela, Hugo Chávez. Dieterich, planteaba que "es un nuevo sistema político, basado en una economía de equivalencias frente a una de mercado, una democracia participativa directa, y una nueva ética".

Esta postura política propone una unión colectiva en base a principios de autogestión, fomenta una sostenibilidad ambiental y trata de encontrar el valor del cambio al valor del uso, también tiene como características como altos valores de solidaridad, fraternidad, defensa en la diferencia de clases. No es una estructura individualista como el capitalismo, al contrario, utiliza una ética colectiva (Fernández, 2014).

El objetivo de esta ideología política se base principalmente en priorizar los social, lo ambiental, respetar el ambiente natural, la vida y calidad humana, tratando de mitigar o erradicar las desigualdades jerárquicas, dejando a un lado el neoliberalismo (Fernández, 2014). Tener soberanía e integración regional contra el funcionamiento capitalista, tratando de erradicar la explotación. donde el pueblo es el que elige a sus lideres y además tomar responsabilidad directa en las decisiones que ellos toman.

El socialismo del siglo XXI en Latinoamérica vino de la mano con el expresidente Hugo Chávez, donde aprobó una constitución con misiones bolivarianas e implemento consejos comunales para incentivar la participación ciudadana directa dentro de su periodo de 2001 a 2007. Este fue el inicio donde se estableció las bases del socialismo en Latinoamérica. En el periodo de 2005 Chávez trajo a la ley de tierras y desarrollo agrario, poniendo a su merced más de 3.5 millones de hectáreas para su distribución, sustentando de que esto ayudaría a la seguridad alimentaria y equidad rural.

Desde sus principios, el capitalismo fue el enemigo y objetivo claro a erradicar, por lo que el socialismo del XXI se destacó del socialismo tradicional gracias a sus valores comunitarios, empatía radical y su cercanía con la soberanía con la ciudadanía. En 2011, Venezuela tuvo la cumbre de la CELAC, donde se instauro la ley “antigolpe” donde se

defendió la soberanía regional donde se armó una red de partidos que tenían las mismas directrices de Chávez en gobiernos como Bolivia, Ecuador.

## Ética y responsabilidad en el uso de IA para análisis social

El uso de la inteligencia artificial tiene una gran responsabilidad ética, debido a que va más allá de la parte técnica, debe incluir el respeto a los derechos humanos y la dignidad de las personas (Hagerty & Rubinov, 2019). En algunos casos se ha podido comprobar que los algoritmos de aprendizaje automático tienen a producir errores y sesgos, en algunos casos discriminatorios y contra éticos. Los investigadores deben controlar y regular adecuadamente estos percances, velando por la transparencia y validez de la verdad durante la aplicación de la IA.

### *Riesgos y cuidados al interpretar opiniones de usuarios.*

Analizar opiniones sobre la percepción ciudadana sobre algún tema, requiere alto criterio que hay que manejarlo con cautela, ya que una mala interpretación puede generar sesgos e interpretaciones erróneas de la verdadera intención del comentario o texto. Un estudio realizado a cerca de la percepción de la justicia en los algoritmos descubrió que las personas tienden a considerar a un sistema como "justo" si este resultado no está en contra de sus principios o no lo ataca directamente o lo beneficia.

### *Uso ético de modelos de lenguaje preentrenados.*

Los grandes modelos ofrecen gran capacidad de respuesta para temas de contexto social, pero así mismo enfrentan desafíos éticos como alucinaciones informativas, sesgos implícitos, amenazas a la privacidad y dificultad en mostrar la información de forma transparente.

### *Protección de datos en investigaciones con scraping o APIs.*

La recolección de datos con el uso de scraping o APIs de fuentes como redes sociales o plataformas como YouTube tiene también consigo varios retos éticos, ya que un dato no solo por ser público no quita la necesidad de consentimiento, no garantiza la seguridad ni la validez de lo que se afirma o se comenta (Harris, 2016). Las políticas de privacidad de la información necesitan transparencia, consentimiento mientras sea posible, anonimización y protección información sensible que se esté recabando.

### *Sesgos algorítmicos y su impacto en investigaciones sociales.*

Los sesgos en los algoritmos que se emplean y que los usa de forma cotidiana, en algunos casos tergiversan los resultados sociales o muestran desigualdades. Esto puede ocurrir en datos históricos, diseño e implementación de modelos o en el mal uso de un sistema (Iwasiński, 2021). Un estudio revelo que uno de cada 3 análisis sociales es afectado por sesgos y del 10% de estos sesgos, fueron impactos críticos con sus consecuencias. Algunas investigaciones incentivan a tener un enfoque que no solo mitigue el sesgo del tema que se esté tratando, sino que también se tome evidencia las desigualdades estructurales para sacar a la luz las desigualdades y así promover y crear políticas justas para estos algoritmos.

## Inteligencia Artificial

Russell y Norvig (2010) definen la IA como "el estudio de agentes que reciben percepciones del entorno y realizan acciones que maximizan sus posibilidades de éxito" (Artificial Intelligence: A Modern Approach, Prentice Hall).

La inteligencia artificial abarca cualquier sistema que simula inteligencia humana ya sea razonamiento, planificación, aprendizaje, precepción o manipulación de objetos.

## Aprendizaje Automático

El aprendizaje automático es una rama de la inteligencia artificial donde los sistemas aprenden de patrones y toman decisiones sin una programación con instrucciones explicitas. Según un estudio en Machine learning HandBook, el aprendizaje automático cubre los algoritmos estadísticos tradicionales hasta redes neuronales, que se usan para tareas como clasificación, regresión y clusterización.

Los enfoques principales son el aprendizaje supervisado y el aprendizaje no supervisado. En el aprendizaje supervisado, se requiere que los textos estén etiquetados antes del entrenamiento, con ello, el modelo a entrenar aprende patrones durante su entrenamiento para después aplicarlos a datos nuevos que el modelo nunca ha visto Por otro lado el aprendizaje no supervisado trata de identificar estructuras en datos sin etiquetas mediante técnicas como clustering o agrupación (Thangaraj & Sivakami, 2018). También se puede hablar de aprendizaje semi-supervisado cunado existe una combinación entre datos etiquetados y no etiquetados, lo cual toma beneficencia de cada uno para una mejor clasificación (Duarte & Berton, 2023).

### *Aprendizaje supervisado*

El aprendizaje supervisado utiliza datos que están previamente etiquetados antes de su procesamiento. Dentro del entrenamiento, el modelo trata de mitigar los errores que se puedan presentar durante las predicciones sobre las etiquetas verdaderas. Sus principales usos son para clasificación y regresión, pero siempre dependiendo de que los datos sean etiquetados.

### *Aprendizaje no supervisado*

El aprendizaje no supervisado se entrena con datos sin etiquetas, buscando patrones y comportamientos entre los datos, como agrupaciones o reglas asociativas. Las técnicas que comúnmente se utilizan para este aprendizaje son, clustering, detección de anomalías y agrupamiento. Este es de gran utilidad para poder explorar y encontrar patrones en los datos.

### *Aprendizaje profundo (DeepLearning)*

Este tipo de aprendizaje utiliza redes neuronales en varias capas para aprender representaciones jerárquicas que los datos puedan tener. Las arquitecturas como CNN, RRN, y Transformers, tienen la capacidad de captar características complejas sin la necesidad de la supervisión humana en la extracción de rasgos. Utilizado en sistemas de visión, voz y texto. Se caracterizan también por su capacidad de procesar alta data mediante el procesamiento con hardware como GPUs o NPUs.

### **Procesamiento de lenguaje Natural (NLP)**

Es un campo de la inteligencia artificial que une la lingüística computacional, estadística y aprendizaje automático para descifrar y poder interpretar el lenguaje humano. Últimamente, gracias al avance y métodos de DeepLearning, se ha podido desarrollar de mejor manera el procesamiento de lenguaje natural. Hoy en día hay modelos avanzados como LSTM, CNN, y Transformers como BERT y GPT que ayudan al desempeño de tareas de traducción, análisis de sentimientos, extracción de información y la extracción del dialogo.

### *Clasificación de texto*

La clasificación de texto es una actividad de procesamiento de lenguaje natural (PLN) o (NLP) donde el objetivo central se basa en asignar de forma automática las etiquetas predefinidas a fragmentos de texto como lo pueden ser (texto, párrafos, documentos, comentarios). organizándolos por temas, sentimientos o intenciones (IBM, s. f.; Minaee et al., 2020).

### *Vectorización y ponderación de términos*

El principal proceso para la clasificación de texto es transformar el contenido lingüístico en vectores numéricos para que los algoritmos lo puedan procesar. Gracias a las técnicas de vectorización, cada documento o texto, tiene unas secuencias de valores, un valor por cada termino. Esto se lo puede realizar con librerías ya establecidas donde se puede ajustar los pesos y el balance de porcentaje a cada palabra según ciertos criterios que son modificables para tener una mejor precisión en los entrenamientos.

***Figura1***

*Vectorización de texto*

| | Gato | Gusta | Mordio | Perro | Queso | Raton |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 1 | 0 | 0 | 1 | 1 |

## Modelos de aprendizaje automático en clasificación de texto

El modelado con aprendizaje automático para clasificar texto incluye texto algoritmos estadísticos simples y también algoritmos avanzados de ensamblaje. Cada uno de ellos debe transformar los textos en vectores y de ahí se empieza a predecir en categorías simples o multiclase.

### *Naive Bayes*

Este es un algoritmo clasificador basado en el teorema de bayes, que da independencia entre características. A pesar de no ser un algoritmo tan complejo, ha demostrado tener una gran capacidad en su rendimiento y desempeño con grandes datos (Raschka, 2014; Shuo Xu, 2016). Jurafsky y Martin (2019) argumentan que es muy efectivo en clasificación de Spam y en clasificación de sentimientos, también tienen un bajo consto computacional y tiene robustez en varios aspectos.

### *Random Forest.*

La función de Random Forest se basa en la combinación de múltiples árboles de decisión entrenados en varios subconjuntos aleatorios, donde se mejora la precisión y se reduce el sobreajuste en datos que tienen alta dimensión (Biau & Scornet, 2016; Jalal et al., 2022). La versión de "semantics-aware" lo hace tener un buen desempeño en textos cortos y con el majeo de ruido y donde hay información dispersa (Jalal et al., 2022).

### *SVM*

El manejo de SVM es en base a clasificación con discriminación las cuales buscan maximizar el margen entre las clases a predecir. Son muy efectivos en clasificación de texto y

en espacios dimensionales altos. También ofrece una máxima capacidad de generalización gracias a su gran separación (Moguerza & Muñoz, 2006).

### *XGBoost*

Es un método boosting con un gran desempeño para problemas de clasificación multiclase. A diferencia de otros, este usa paralelismo y su rendimiento es superior en competiciones.

### *Preprocesamiento necesario: vectorización, TF-IDF, embeddings.*

El preprocesamiento se reviere a la conversión de texto en vectores numéricos para poder entrenar a los modelos, la vectorización convierte documentos o texto en vectores, generalmente usando TF-IDF o Bag-of-words (Manning et al., 2008). TF-IDF, ayuda a clasificar cuán importante es un término en un texto, para así mejorar la discriminación y poder atenuar palabras frecuentes (Manning et al., 2008; Aizawa, 2003). Por otro lado, Embeddings ayuda a representar palabras o textos en vectores densos y semánticos como Word2Vec, GloVe o Bert, ayudando a capturar relaciones contextuales avanzadas.

## Modelos de lenguaje a gran escala (LLMs)

### *LLMs*

Los modelos de lenguajes basados en la arquitectura Transformers, entrenados con una cantidad inmensa de datos, miles, millones o trillones de parámetros. Se caracterizan por su gran capacidad para entender cualquier tipo de lenguaje humano sin necesidad de algún ajuste adicional (Minaee et al., 2024; Hadi et al., 2023).

### *ChatGpt*

Fue desarrollado por OpenAI, usa Transformers y se entrena con aprendizaje no supervisado y ajuste con el feedback humano, es decir se va entrenando con la interacción con el humano (SFT, RLHF). Con un despliegue en aplicaciones de escritura académica, programación, generación de videos, imágenes y más (Kalyan, 2023; BioData Mining, 2023).

### *DeepSeek*

Es una empresa China que saco a despliegue modelo de LLM de código abierto. Su arquitectura es de tipo Mixture-of-Experts (MoE) y capacitación eficiencia. El modelo DeepSeeK-R1 (671 B) tiene un rendimiento similar a GPT-4. Tiene una escala eficiente y transparente, es un modelo muy influyente hoy en día dentro del entorno open source.

## Métricas de evaluación

### *Accuracy*

La métrica de evaluación accuracy mide la proporción de predicciones correctas ya sean (POS, NEG o NEU) respecto al total de casos. No puede ser tan buena métrica en casos donde haya gran desbalance de datos, porque puede presentar valores altos por la clase mayoritaria dejando a un lado las clases menores.

### *Precision*

Indica la proporción de verdaderos positivos sobre todas las predicciones positivas.

### *Recall*

Sensibilidad, mide la proporción de verdaderos positivos sobre todas las instancias reales positivas.

***F1-score.***

Es la media armónica de la precisión y recall, Es de gran ayuda en escenarios con clases desbalanceadas.

***Índice Gini***

Se para determinar la impureza de un nodo en arboles de decisión para elegirse por la capacidad para poder medir la desigualdad en la distribución entre clases. Cuan menor es el índice de Gini más pura es la división.

***Log loss***

La métrica de los loss mide la desviación entre las probabilidades que se pronosticaron y las que se observaron. Se penaliza fuertemente la confianza en predicciones incorrectas.

***Rango de distancias al hiperplano***

Se usa en SVM, es la distancia minina del hiperplano a los vectores de soporte. Un margen amplio mejora la generalización del modelo.

***Matriz de confusión.***

La matriz de confusión muestra el conteo de los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. Para así poder calcular las métricas anteriores.

***Ajustar hiperparámetros***

Determinación de valores óptimos en parámetros para un mejor rendimiento del modelo. Si se realiza una buena configuración se puede mejorar el rendimiento y evita el sobreajuste.

### *Validación cruzada*

La validación cruzada y la división entre el entrenamiento, la validación y la prueba. Ayudan a tener observar la capacidad de generalización del modelo sin algún sesgo.

### *GridSearch*

Es una técnica que explora algunas posibles combinaciones que se puede tener en los hiperparámetros, usando validación cruzada para encontrar la mejorar combinación para el modelo.

## Metodología CRISP-DM

El termino CRISP-DM, Cross-Industry Standard Process for Data Mining) es una metodología que desarrollada por IBM para gestionar proyectos de minería de datos. Se compone de 6 fases, estas 6 fases son iterativas, no es un proceso lineal. Sirve para alinear los objetivos del negocio con los objetivos de minería.

### *Descripción de sus seis fases.*

1. **Comprensión del negocio (Business Understanding)**

Se inicia definiendo los objetivos empresariales, las metas del proyecto y los criterios de éxito. Esta fase establece el rumbo alineado con las necesidades reales de la organización

**Tópicos**:

- Determinar los objetivos del negocio.
- Evaluar la situación actual (recursos, limitaciones, riesgos).

- Formular los objetivos de minería de datos.
- Redactar un plan de proyecto inicial.

**Fuente:** IBM SPSS Modeler Documentation (s. f.); Data Science PM (2024).

## 2. Comprensión de los datos (Data Understanding)

Consiste en recopilar datos preliminares, explorar sus características, evaluar su calidad y documentar posibles problemas. Esta etapa permite seleccionar fuentes relevantes y detectar errores tempranamente.

**Tópicos**:

- Recolección de datos inicial.
- Descripción de datos (tipo, cantidad, formato).
- Exploración de los datos (visualización, estadística).
- Verificación de la calidad de los datos.

**Fuente:** IBM CRISP-DM Overview; Smart Vision Europe (2022).

## 3. Preparación de datos (Data Preparation)

Popularmente conocida como data wrangling, implica selección, limpieza, creación de variables, integración y transformación, para construir el dataset final que se utilizará en modelado

**Tópicos**:

- Selección de variables relevantes.
- Limpieza de datos (valores faltantes, duplicados).
- Construcción de atributos nuevos.
- Integración de datos.
- Formateo de datos.

**Fuente:** IBM SPSS Modeler; Business & Decision (2023).

## 4. Modelado (Modeling)

Aquí se seleccionan algoritmos adecuados (como clasificación o regresión), se formulan los conjuntos de training/test, y se construyen modelos ajustados, iterando hasta obtener un rendimiento satisfactorio.

**Tópicos**:

- Selección del modelo.
- División de datos.
- Entrenamiento.
- Evaluación.

**Fuente:** IBM CRISP-DM Help; Kohavi (1995).

## 5. Evaluación (Evaluation)

Valida si los modelos cumplen los objetivos del negocio, revisa la integridad del proceso y decide si están listos para desplegar o requieren ajustes adicionales.

**Tópicos**:

- Interpretación de los resultados del modelo.
- Revisión del cumplimiento de objetivos.
- Validación contra criterios del negocio.
- Decisión de continuar, repetir fases o ajustar.

**Fuente:** DataScience-PM.com; IBM Documentation.

## 6. Implementación (Deployment)

Se implementa la solución en producción (informe, integración en sistemas, monitoreo, mantenimiento). Incluye planificación de despliegue y seguimiento periódico.

**Tópicos**:

- Generación de informes y visualizaciones.
- Creación de sistemas automatizados o dashboards.
- Entrenamiento de usuarios o stakeholders.
- Mantenimiento y monitoreo continuo.
- Documentación final del proyecto.

**Fuente:** IBM SPSS Modeler Documentation; PSE Consultoría (2022).

# CAPÍTULO III METODOLOGÍA

## Investigación científica

La presente investigación se desarrolló dentro de un enfoque cuantitativo y aplicado, experimental debido a que se buscó analizar un fenómeno social de gran relevancia por medio de datos reales obtenidos de plataformas digitales como videos de YouTube sobre el socialismo en América Latina. A través del análisis de estos comentarios de varios videos relevantes al tema, se pretendió observar y medir la perspectiva de la ciudadanía sobre esta ideología política en aspectos geográficos y temporales.

Desde la perspectiva científica, se partió de la observación sistemática del comportamiento discursivo en entornos digitales, considerando a las redes sociales y al internet como espacios de expresión pública en los que se evidencian posturas emocionales y valoraciones políticas de los usuarios. A partir de esta observación, se procedió a un proceso experimental que consiste en el diseño, entrenamiento y evaluación de modelos de análisis de sentimientos, con el objetivo de clasificar los comentarios en categorías como positivos, negativos o neutros. Esta experimentación no solo incluye algoritmos tradicionales de aprendizaje automático, sino también modelos avanzados de inteligencia artificial como los LLMs (Modelos de Lenguaje a Gran Escala), entre ellos ChatGPT o DeepSeek, a los cuales se les suministra el mismo conjunto de comentarios para evaluar su rendimiento en tareas de clasificación emocional.

La investigación fue aplicada en tanto que no se limita a la observación pasiva del fenómeno, sino que tuvo un propósito concreto: desarrollar una solución analítica que permita entender cómo se ha construido, transformado y polarizado la opinión pública frente al socialismo en la región. En este sentido, el estudio articuló la teoría social y política con

herramientas de ciencia de datos e inteligencia artificial, permitiendo abordar una problemática actual desde un enfoque interdisciplinario y riguroso.

**Investigación técnica**

Desde el punto de vista técnico, la investigación adoptó como referencia la metodología CRISP-DM (Cross Industry Standard Process for Data Mining), una de las más utilizadas y reconocidas en proyectos de minería de datos. Esta metodología proporciona una estructura flexible y cíclica que permite abordar el análisis de datos de manera ordenada y eficiente. En este estudio, CRISP-DM se implementó solo como referencia, iniciando con la comprensión del contexto político y social en el que se inscribe el fenómeno del Socialismo del Siglo XXI, así como con la identificación de los objetivos específicos de la investigación relacionados con la percepción ciudadana en redes sociales.

Una vez definido el propósito del estudio, se pasó a la exploración y comprensión de los datos disponibles, en este caso, comentarios de usuarios en plataformas como YouTube. Posteriormente, se realizó la recolección de los datos mediante el uso de APIs públicas, seguido de un proceso de preparación que incluye la limpieza, transformación y normalización del texto para su posterior análisis. Esta etapa fue fundamental para garantizar la calidad del modelado.

En la fase de modelado, se aplicaron tanto algoritmos tradicionales de aprendizaje automático, como también modelos avanzados basados en inteligencia artificial mediante el uso de APIs de modelos de lenguaje a gran escala, como ChatGPT o DeepSeek. El objetivo fue comparar el rendimiento de ambas aproximaciones para determinar cuál ofrece mejores resultados en la tarea de clasificación de sentimientos.

Finalmente, se evaluó el desempeño de los modelos utilizando métricas como precisión, recall, F1-score y exactitud. Esta evaluación permitió establecer conclusiones sobre

la efectividad de cada enfoque y contribuye a la validación de los resultados obtenidos. En conjunto, con la aplicación como referencia de la metodología CRISP-DM asegura un desarrollo técnico estructurado, sistemático y alineado con los estándares actuales de la ciencia de datos.

# CAPÍTULO IV: PERCEPCIONES CIUDADANAS SOBRE EL SOCIALISMO EN LATINOAMÉRICA EN LOS ÚLTIMOS DIEZ AÑOS MEDIANTE TÉCNICAS DE MINERÍA DE TEXTO Y ANÁLISIS DE SENTIMIENTO APLICADOS A COMENTARIOS EXTRAÍDOS DE REDES SOCIALES.

## Introducción

El siguiente capítulo propone el proceso de análisis de sentimientos aplicado a comentarios en videos de YouTube relacionados con el Socialismo del Siglo XXI en Latinoamérica. Para este análisis, se utilizó la metodología CRISP-DM (Cross Industry Standard Process for Data Mining), donde se estructuró el proyecto, desde la fase 1, comprensión del problema, hasta la fase 5, evaluación de los modelos predictivos. Esta metodología facilito el análisis de forma sistemática sobre la percepción ciudadana, mediante la comparación de modelos clásicos de aprendizaje automático y de dos modelos de inteligencia artificial basado en lenguaje (LLM) para la clasificación de emociones.

## 4.1 Extracción de comentarios relacionados al socialismo en Latinoamérica en redes sociales.

### FASE 1: Comprensión del problema

Para iniciar con el análisis de percepción ciudadana, se analizó el contexto político y social del Socialismo del Siglo XXI en Latinoamérica y cómo este ha generado posturas bien divididas en redes sociales sobre el tema. Esta etapa permitió identificar la necesidad de una herramienta automatizada que clasifique las emociones de las personas sobre este tema en categorías: positiva, negativa o neutra, proporcionando una visión más objetiva de la percepción colectiva.

### FASE 2: Comprensión de los datos

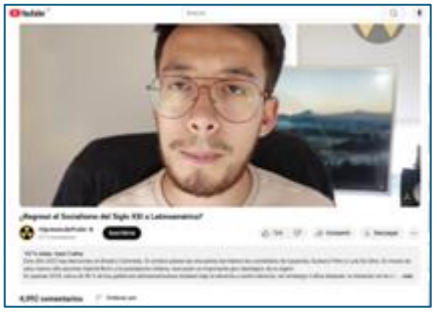
Para cumplir este propósito, se seleccionaron comentarios desde la herramienta de videos "YouTube" que topan temas relacionados con el socialismo en países como Venezuela, Bolivia, Ecuador y otros. Se escogieron los videos controversiales, más representativos, más recientes, con gran número de visualizaciones y con buena participación ciudadana.

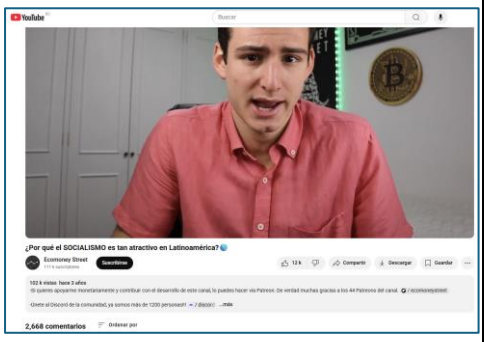
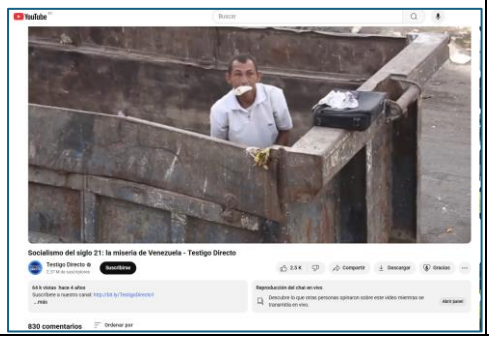
A continuación, se presenta la tabla de los videos que fueron seleccionados:

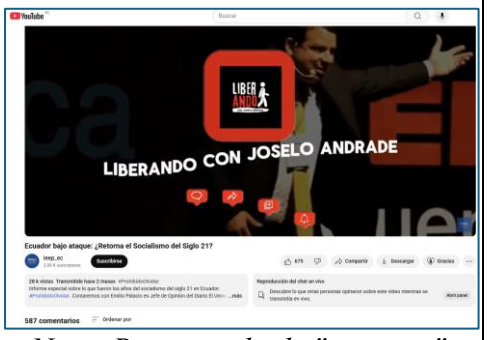
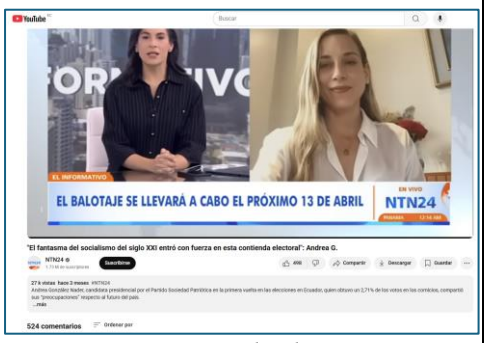
***Recopilar datos iniciales:***

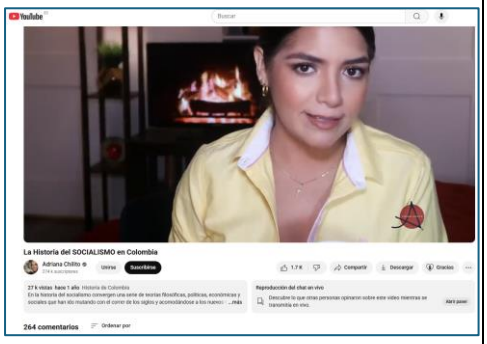
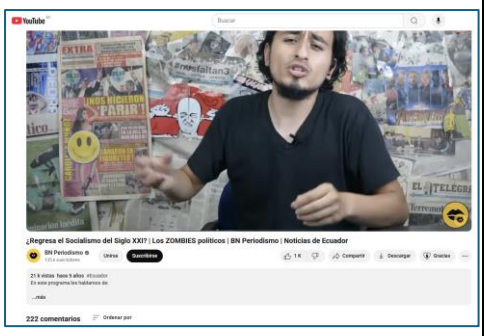
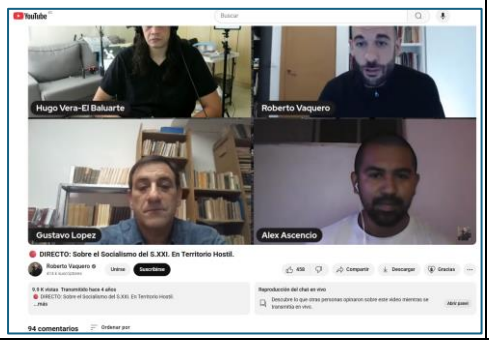
**Tabla de videos para el análisis**

***Tabla 1***
*Tabla de videos recolectados*

| *Nombre* | *Video* | *Url* |
|---|---|---|
| El socialismo del siglo XXI acabó con BoliviaBO ¿Ecuador es el siguiente? EC | ***Figura 2***<br>*Pantalla video 1*<br>  <br>*Nota: Recuperado de "DESDE MI PUNTO DE VISTA por Constantino de Miguel"* | https://www.youtube.com/watch?v=83LCyPXnbVA&t=33s |
| ¿Regresó el Socialismo del Siglo XXI a Latinoamérica? | ***Figura 3***<br>*Pantalla video 2*<br>  <br>*Nota: Recuperado de "Hipotesis de Poder"* | https://www.youtube.com/watch?v=1LwQCiwue3g&t=21s |

| | | |
|---|---|---|
| ¿Por qué el SOCIALISMO es tan atractivo en Latinoamérica? | ***Figura 4***<br>*Pantalla video 3*<br>  <br><br>*Nota: Recuperado de " Ecomoney Street"* | https://www.youtube.com/watch?v=npn8r7wh9Ko |
| POR QUÉ EL SOCIALISMO TRIUNFA EN LATINOAMÉRICA | ***Figura5***<br>Pantalla video 4<br>  <br><br>*Nota: Recuperado de " CRITICALANDIA"* | https://www.youtube.com/watch?v=62tda2CkQ8s |
| Socialismo del siglo 21: la miseria de Venezuela - Testigo Directo | ***Figura6***<br>*Pantalla video 5*<br>  <br> | https://www.youtube.com/watch?v=k14KVEBCx50 |

| | | |
|---|---|---|
| | *Nota: Recuperado de " Testigo Directo"* | |
| Rafael Correa sobre el Materialismo Dialéctico, Marxismo , Socialismo del siglo XXI | ***Figura7***<br>*Pantalla video 6*<br>  <br><br>Nota: Recuperado de " VHGKASPAROV | https://www.youtube.com/watch?v=y5VmGaOUojM |
| Ecuador bajo ataque: ¿Retorna el Socialismo del Siglo 21? | ***Figura8***<br>*Pantalla video 7*<br>  <br><br>*Nota: Recuperado de " ieep_ec"* | https://www.youtube.com/watch?v=nC-AK7bhUfc |
| "El fantasma del socialismo del siglo XXI entró con fuerza en esta contienda electoral": Andrea G. | ***Figura9***<br>Pantalla video 8<br>  <br><br>*Nota: Recuperado de "NTN24"* | https://www.youtube.com/watch?v=TjpfJGGuwTc |

| | | |
|---|---|---|
| La Historia del SOCIALISMO en Colombia | ***Figura10***<br>*Pantalla video 9*<br>  <br>*Nota: Recuperado de " Adriana Chilito"* | https://www.youtube.com/watch?v=A8nq_Q2_KBY |
| ¿Regresa el Socialismo del Siglo XXI? \| Los ZOMBIES políticos \| BN Periodismo \| Noticias de Ecuador | ***Figura11***<br>*Pantalla video 10*<br>  <br>*Nota: Recuperado de " BN Periodismo"* | https://www.youtube.com/watch?v=HZa5voSUlS0 |
| DIRECTO: Sobre el Socialismo del S.XXI. En Territorio Hostil. | ***Figura 12***<br>*Pantalla video 11*<br>  | https://www.youtube.com/watch?v=g3OtscDTzNc |

| | | |
|---|---|---|
| | *Nota: Recuperado de " Roberto Vaquero"* | |
| Ecuador bajo ataque: Retornará el Socialismo del siglo 21 - Una mirada desde lo económico | ***Figura13***<br>*Pantalla video 12*<br>  <br><br>*Nota: Recuperado de "ieep_ec"* | https://www.youtube.com/watch?v=yP51AWdcmLI |
| Ecuador bajo ataque: ¿Retornará el Socialismo del siglo XXI? - Más de un intento de desdolarizar | ***Figura14***<br>*Pantalla video 13*<br>  <br><br>*Nota: Recuperado de "ieep_ec"* | https://www.youtube.com/watch?v=CHsGEOw2PlY |
| Bolivia sigue sometida al socialismo del siglo XXI o castrochavismo | ***Figura 15***<br>*Pantalla video 14*<br>  <br><br>*Nota: Recuperado " Carlos Sanchez Berzaín"* | https://www.youtube.com/watch?v=l8XU2_2B298 |

| | | |
|---|---|---|
| EL FUTURO ECONÓMICO DEL ECUADOR SI GANA LUISA GONZÁLEZ: ¿REGRESA EL SOCIALISMO DEL SIGLO 21? | ***Figura16***<br>*Pantalla video 15*<br>  <br>*Nota: Recuperado de "Libre Opinión Ecuador"* | https://www.youtube.com/watch?v=yxsSVkEopEs |
| Crisis Económica en Bolivia: El Espejo de Venezuela 🚨 Sin Dólares ni Comida: el Fracaso Socialista | ***Figura17***<br>*Pantalla video 16*<br>  <br>*Nota: Recuperado de "Gabo Clips"* | https://www.youtube.com/watch?v=48p81I_We_8 |
| Crisis Económica en Bolivia: Cómo Evo Morales Copió el Fracaso de | ***Figura 18***<br>*Pantalla video 17* | https://www.youtube.com/watch?v=VgSJ7JS720c |

| | | |
|---|---|---|
| Venezuela y el<br><br>Modelo Socialista |  <br>*Nota: Recuperado de "Gabo Clips"* | |

*Nota: Tabla con los videos que se van a utilizar para el análisis de percepción del socialismo del siglo XXI*

***Obtención de clave ApiKey YouTube Data V3***

Se utilizó la API de YouTube Data V3 por medio de la plataforma Google Cloud, para obtener los comentarios de los comentarios previamente seleccionados. Esta extracción se realizó con Python utilizando la librería GoogleApiClient.la cual permitió construir y enviar solicitudes a los servicios de GoogleCloud, por medio del ID único de cada video. Para ello se hicieron varias llamadas a la API donde se aplicó técnicas de solicitudes por medio de API REST dentro de un bucle para hacer varias llamadas tomando en cada iteración el ID del video. Estos comentarios se los guardó en una estructura de datos tipo “lista de diccionarios”, para después transformarlos en un DataFrame con Pandas. Finalmente se guardó este DataSet en un CSV llamado “yotube_comments-csv”. (Ver anexo - procedimiento para descargar comentarios de YouTube).

***Describir los Datos***

Se obtuvo un DataSet con los siguientes atributos:

***Tabla 2***
*Tabla de atributos del DataSet*

| ***Nombre*** | ***Descripción*** | ***Tipo de campo*** |
|---|---|---|
| ***video_id*** | Id del video descargado | Categórico nominal: Identificador único, sin orden inherente. |
| ***autor*** | Nombre del usuario que realizo ese comentario | Categórico nominal: Nombre sin orden lógico. |
| ***text*** | Comentario realizado por el usuario en el video | Categórico nominal (texto libre) |
| ***published_at*** | Fecha de publicación exacta del comentario | Cuantitativo continuo (temporal): se mide en unidades de tiempo continuas. |
| ***like_count*** | Recuento de likes adquirido por ese comentario | Cuantitativo discreto: Números enteros. |

**Dimensiones del DataSet:**

Una vez finalizado el proceso de extracción de comentarios de YouTube, se realizó una exploración en los datos. Como se muestra en la figura 18, el DataSet contuvo 6826 Filas y 5 columnas previamente mencionadas.

***Figura 19***

*Dimensiones del DataSet youtube_comments.csv*

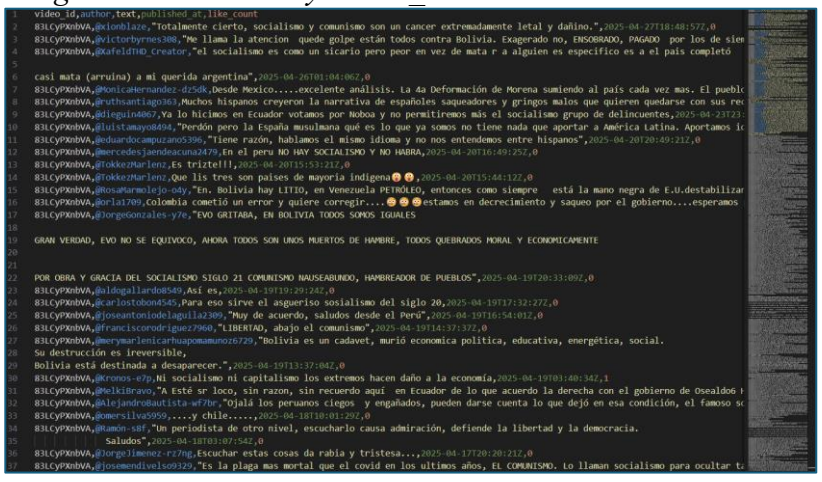
```
    # Obtener las dimensiones del dataset
    filas, columnas = df.shape

    print(f"El dataset tiene {filas} filas y {columnas} columnas.")
[5] ✓ 0.0s

··  El dataset tiene 6826 filas y 5 columnas.
```

***Explorar datos***

Se procedió con una exploración preliminar de los primeros registros del DataSet. Este primer contacto con los datos ayudó a confirmar que los comentarios hablen sobre el tema propuesto y que estos datos sean relevantes para el análisis de sentimientos.

***Figura 20***

*Pantalla primeros registros del DataSet youtube_comments.csv*



```
video_id,author,text,published_at,like_count
83LCyPXnbVA,@xlonblaze,"Totalmente cierto, socialismo y comunismo son un cancer extremadamente letal y dañino.",2025-04-27T18:48:57Z,0
83LCyPXnbVA,@victorbyrnes308,"Me llama la atencion  quede golpe están todos contra Bolivia. Exagerado no, ENSOBRADO, PAGADO  por los de sien
83LCyPXnbVA,@XafeldTHD_Creator,"El socialismo es como un sicario pero peor en vez de mata r a alguien es especifico es a el pais completo

casi mata (arruina) a mi querida argentina",2025-04-26T01:04:06Z,0
83LCyPXnbVA,@MonicaHernandez-dz5dk,Desde Mexico.....excelente análisis. La 4a Deformación de Morena sumiendo al país cada vez mas. El pueblo
83LCyPXnbVA,@ruthsantiago363,Muchos hispanos creyeron la narrativa de españoles saqueadores y gringos malos que quieren quedarse con sus rec
83LCyPXnbVA,@dieguin4067,Ya lo hicimos en Ecuador votamos por Noboa y no permitiremos más el socialismo grupo de delincuentes,2025-04-23T23:
83LCyPXnbVA,@luistamayo8494,"Perdón pero la España musulmana qué es lo que ya somos no tiene nada que aportar a América Latina. Aportamos ic
83LCyPXnbVA,@eduardocampuzano5396,"Tiene razón, hablamos el mismo idioma y no nos entendemos entre hispanos",2025-04-20T20:49:21Z,0
83LCyPXnbVA,@mercedesjaendeacuna2479,En el peru NO HAY SOCIALISMO Y NO HABRA,2025-04-20T16:49:25Z,0
83LCyPXnbVA,@TokkezMarlenz,Es triztel!!,2025-04-20T15:53:21Z,0
83LCyPXnbVA,@TokkezMarlenz,Que lis tres son paises de mayoria indigena😡😡,2025-04-20T15:44:12Z,0
83LCyPXnbVA,@RosaMarmolejo-o4y,"En. Bolivia hay LITIO, en Venezuela PETRÓLEO, entonces como siempre  está la mano negra de E.U.destabilizar
83LCyPXnbVA,@orla1709,Colombia cometió un error y quiere corregir....😡😡😡estamos en decrecimiento y saqueo por el gobierno....esperamos
83LCyPXnbVA,@JorgeGonzales-y7e,"EVO GRITABA, EN BOLIVIA TODOS SOMOS IGUALES

GRAN VERDAD, EVO NO SE EQUIVOCO, AHORA TODOS SON UNOS MUERTOS DE HAMBRE, TODOS QUEBRADOS MORAL Y ECONOMICAMENTE


POR OBRA Y GRACIA DEL SOCIALISMO SIGLO 21 COMUNISMO NAUSEABUNDO, HAMBREADOR DE PUEBLOS",2025-04-19T20:33:09Z,0
83LCyPXnbVA,@aldogallardo8549,Así es,2025-04-19T19:29:24Z,0
83LCyPXnbVA,@carlostobon4545,Para eso sirve el asqueriso sosialismo del siglo 20,2025-04-19T17:32:27Z,0
83LCyPXnbVA,@joseantoniodelaguila2309,"Muy de acuerdo, saludos desde el Perú",2025-04-19T16:54:01Z,0
83LCyPXnbVA,@franciscorodriguez7960,"LIBERTAD, abajo el comunismo",2025-04-19T14:37:37Z,0
83LCyPXnbVA,@merymarlenicarhuapomamunoz6729,"Bolivia es un cadavet, murió economica politica, educativa, energética, social.
Su destrucción es ireversible,
Bolivia está destinada a desaparecer.",2025-04-19T13:37:04Z,0
83LCyPXnbVA,@Kronos-e7p,Ni socialismo ni capitalismo los extremos hacen daño a la economía,2025-04-19T03:40:34Z,1
83LCyPXnbVA,@MelkiBravo,"A Esté sr loco, sin razon, sin recuerdo aquí  en Ecuador de lo que acuerdo la derecha con el gobierno de Osealdos
83LCyPXnbVA,@AlejandroBautista-wf7br,"ojalá los peruanos ciegos  y engañados, pueden darse cuenta lo que dejó en esa condición, el famoso so
83LCyPXnbVA,@omersilva5959,...y chile.....,2025-04-18T18:01:29Z,0
83LCyPXnbVA,@ramon-s8f,"Un periodista de otro nivel, escucharlo causa admiración, defiende la libertad y la democracia.
Saludos",2025-04-18T03:07:54Z,0
83LCyPXnbVA,@JorgeJimenez-rz7ng,Escuchar estas cosas da rabia y tristesa...,2025-04-17T20:20:21Z,0
83LCyPXnbVA,@josemendivelso9329,"Es la plaga mas mortal que el covid en los ultimos años, EL COMUNISMO. Lo llaman socialismo para ocultar t
```

**Longitud de comentarios del DataSet:**

También como parte de la exploración de datos, se revisó la longitud de comentarios descargados y la longitud de solo las palabras para ver la variabilidad entre comentarios. Para esto, se generó la estadística descriptiva con la función describe ().

***Figura 21***

*Verificar dimensión de los comentarios*

```
Distribución de la longitud de los comentarios:
count    6833.000000
mean      186.722084
std       293.705752
min         1.000000
25%        54.000000
50%       105.000000
75%       208.000000
max      6159.000000
Name: comment_length, dtype: float64
```

**Longitud de los comentarios solo con palabras contadas.**

***Figura 22***

*Longitud de comentarios conteo de palabras*

```
#Longitud del texto en número de palabras
df['longitud'] = df['text'].apply(lambda x: len(x.split()))
print(df['longitud'].describe())
```

✓ 0.0s

```
count    6833.000000
mean       31.714767
std        49.284499
min         1.000000
25%         9.000000
50%        18.000000
75%        35.000000
max      1030.000000
Name: longitud, dtype: float64
```

**Top comentarios más largos:**

Se identificó los comentarios con mayor número de palabras, se ordenó el DataSet en forma descendente en función de la longitud. En la figura 22 se presenta los 5 comentarios más largos del DataSet.

***Figura 23***

*Top comentarios más largos*

```
# Top más largos
print(df.sort_values('longitud', ascending=False)[['text', 'longitud']].head(5))
```

✓ 0.0s Python

| | text | longitud |
|---|---|---|
| 4591 | No has entendido nada de lo que yo he escrito ... | 1030 |
| 4548 | NO HAY PRUEBAS CONTRA CORREA? \n¡ASÍ ORGANIZAR... | 897 |
| 1038 | *Gran artículo.*\n\n *¿POR QUÉ ES IBEROAMÉRIC... | 755 |
| 2302 | En Colombia el uribismo va para 20 años de est... | 724 |
| 4093 | Soy venezolano y en mi país nunca hubo marxis... | 683 |

**Top comentarios más cortos:**

También se consideró mostrar los 5 primeros comentarios con menor cantidad de palabras como se presenta en la figura 23.

***Figura 24***

*Top comentarios más cortos*

```
# Top más cortos (mayores a 1 palabra)
print(df[df['longitud'] > 1].sort_values('longitud')[['text', 'longitud']].head(5
```

✓ 0.0s Python

| | text | longitud |
|---|---|---|
| 1630 | ¡Buen video! | 2 |
| 6138 | Noboa presidente | 2 |
| 3762 | La verdad | 2 |
| 1042 | Moriremos todos | 2 |
| 3809 | Buen video | 2 |

***Verificar la calidad de los datos:***

**Valores nulos:**

Dentro del DataSet se encontró valores nulos en el autor del comentario, cosa que es irrelevante para nuestro estudio.

***Figura 25***

*Verificación de valores nulos*

```
# Verificar valores nulos
print("\nValores nulos por columna:")
print(df.isnull().sum())
```

✓ 0.0s

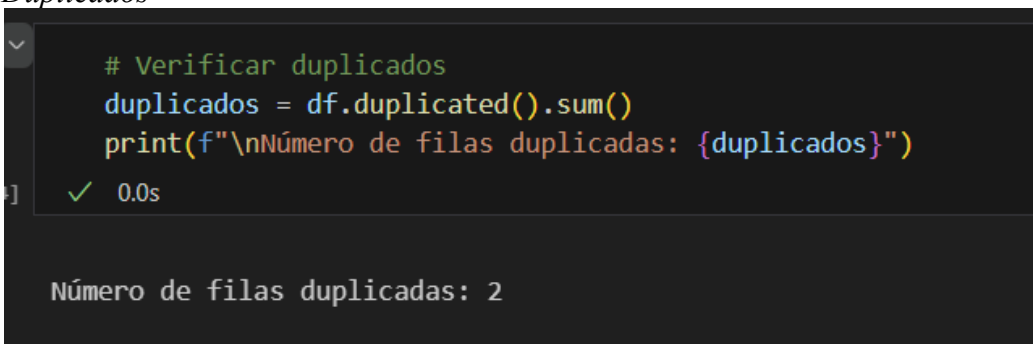
```
Valores nulos por columna:
video_id           0
author            10
text               0
published_at       0
like_count         0
comment_length     0
longitud           0
dtype: int64
```

**Valores duplicados:**

Durante la limpieza del DataSet, se identificó la existencia de comentarios duplicados, por lo que se aplicó la función para detectar los mismos registros en todas las columnas del DataFrame, como se muestra en la figura 25.

***Figura 26***

*Valores Duplicados*

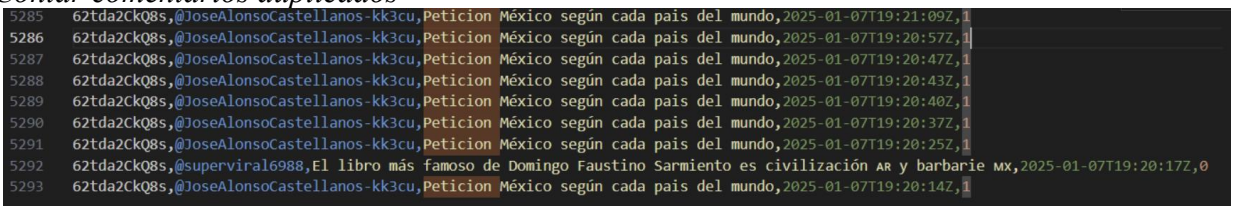



**Contar comentarios duplicados:**

Adicionalmente se hizo una revisión de comentarios duplicados solo en la columna "text" y se revisó de forma manual la duplicidad de los comentarios.

***Figura 27***

*Contar comentarios duplicados*

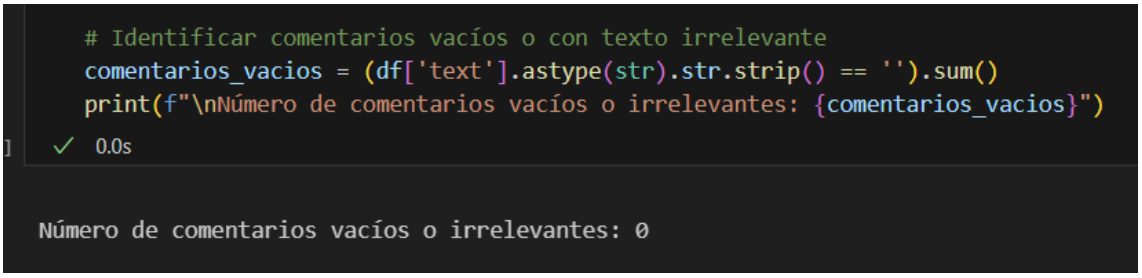



**Comentarios vacíos o irrelevantes:**

Asimismo, se identificó si había comentarios vacíos o sin relevancia para el modelado.

***Figura 28***

*Valores vacíos o irrelevantes*




## 4.2 Construcción y entrenamiento de un modelo de aprendizaje automático

### FASE 3: Preparación de los datos

#### *Seleccionar datos*

Para no modificar la columna original "text", se creó una variable llamada "texto_limpio" que fue de utilidad para la etapa de visualización. En esta columna se duplico los mismos registros que tenía la columna "text" para poder realizar limpieza y transformación de los datos.

***Figura 29***

*Creación de la variable "texto_limpio"*


#### *Limpiar los datos*

En el tópico de limpieza de datos se aplicó técnicas de preprocesamiento de lenguaje natural (NLP) para de esta forma poder limpiar y eliminar dentro de los comentarios caracteres irrelevantes que puedan interferir en la etapa de visualización. (Ver Anexo – Función de

limpieza de texto para visualización). Se aplicó los siguientes procedimientos para la limpieza de datos:

- Conversión a minúsculas.
- Eliminación de tildes.
- Filtrado de Links URls.
- Menciones a usuarios.
- Hashtags.
- Eliminación signos puntación, caracteres especiales.
- Eliminación de emojis.
- Eliminación de espacios innecesarios.
- Filtrado de StopWords.

Gracias a este preprocesamiento del texto se obtuvo una limpieza y depuración del texto original, se obtuvo un texto apto para la etapa de visualización.

**Eliminación de registros duplicados y vacíos:**

Una vez se limpió los datos de los comentarios, se realizó una nueva revisión, pero ahora en la columna "texto_limpio", donde se observó que había 155 filas duplicadas en consecuencia de eliminar StopWords, quitar emojis y demás caracteres. Estos registros se los elimino mediante la función drop duplicates de pandas.

***Figura 30***
*Número de registros duplicados después de la limpieza "texto_limpio"*

```
# Contar cuántos valores duplicados hay en la columna 'texto_limpio'
duplicados_text = df['texto_limpio'].duplicated().sum()
print(f"\nNúmero de filas duplicadas en 'texto_limpio': {duplicados_text}")
✓ 0.0s

Número de filas duplicadas en 'texto_limpio': 155
```

***Figura 31***

*Borrado de datos duplicados o vacíos en "texto_limpio"*

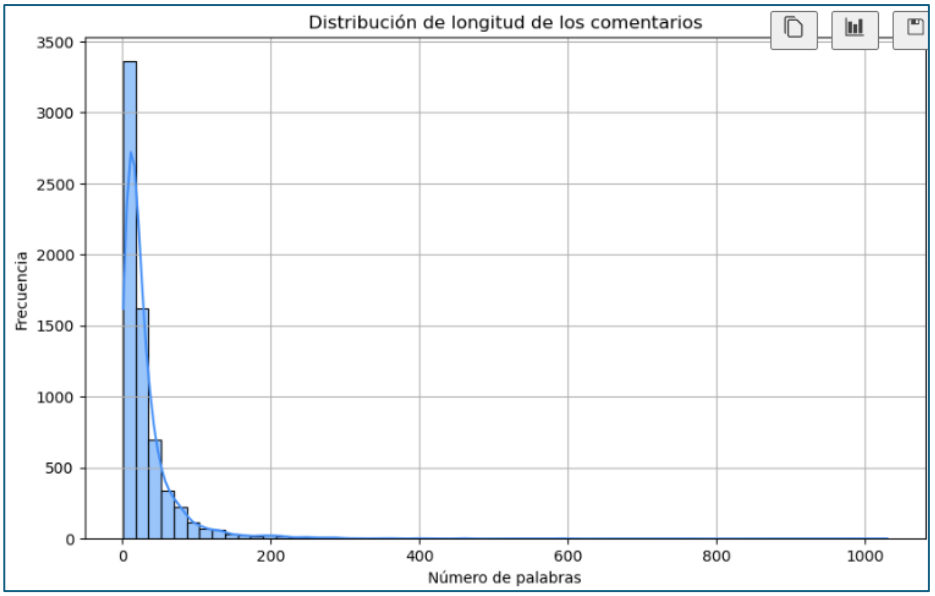
```
#Elimina comentarios duplicados
df = df.drop_duplicates(subset='texto_limpio', keep='first')
✓ 0.0s


#Eliminar campos vacios
df = df[df['texto_limpio'].str.strip() != '']
✓ 0.0s
```

**Visualización de la distribución de la longitud de comentarios**

Para tener más claridad de en los datos y como estos quedaron después de la limpieza de datos, se generó un gráfico de distribución de frecuencias según la cantidad de palabras por comentario. Gracias a este gráfico se pudo observar que hubo más comentarios cortos que largos y que la mayoría oscilan entre las 10 y 50 palabras, también que solo unos pocos comentarios pasaron las 200 palabras u otros pasaron de 500 palabras.

***Figura 32***

*Gráfico distribución de longitud de los comentarios.*




**Nube de palabras:**

Se generó una nube de palabras para identificar visualmente lo que la gente comenta con más frecuencia. Esto fue útil para ver la importancia en cada palabra según su frecuencia, esta es más grande. En la figura 32 se muestran palabras como “izquierda”, “socialismo”, “derecha”, “gobierno”, “país”, “gente” y “socialista”. Palabras que ayudaron a confirmar y a sustentar que los comentarios tienen concordancia con el tema de la percepción del socialismo del siglo XXI. También se observó palabras de países como “Ecuador”, “Venezuela”, “Colombia” y “Bolivia”, lo que ayudó a ver que el contexto se mantiene a nivel latinoamericano.

***Figura 33***

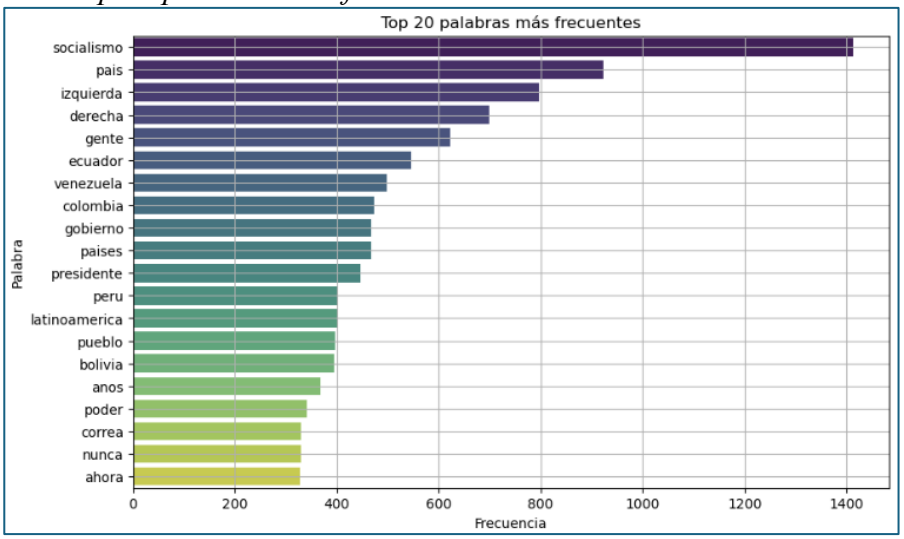
*Gráfico nube de palabras*


**Top 20 palabras más frecuentes.**

Para complementar la nube de palabras, se construyó un gráfico de barras para ver las palabras más frecuentes en orden descendente.

***Figura 34***

*Gráfico de barras Top 20 palabras más frecuentes*



**Binas y trinas en comentarios:**

Para identificar combinaciones más relevantes o frecuentes dentro de los comentarios, se hizo un análisis en los n-gramas más frecuentes en binas y trinas. Esto permitió detectar patrones frecuentes y no solo un análisis por palabras aisladas. En la tabla de la figura 34 se pudo apreciar que los n-gramas frecuentes van acorde al tema, palabras como “socialismo siglo xxi”, “siglo xxi”, “izquierda derecha”, “gobiernos izquierda”, o con nombre de figuras políticas como “daniel noboa”, “rafael correa” o “evo morales”

***Figura 35***

*Tabla de frecuencia de binas y trinas en comentarios*

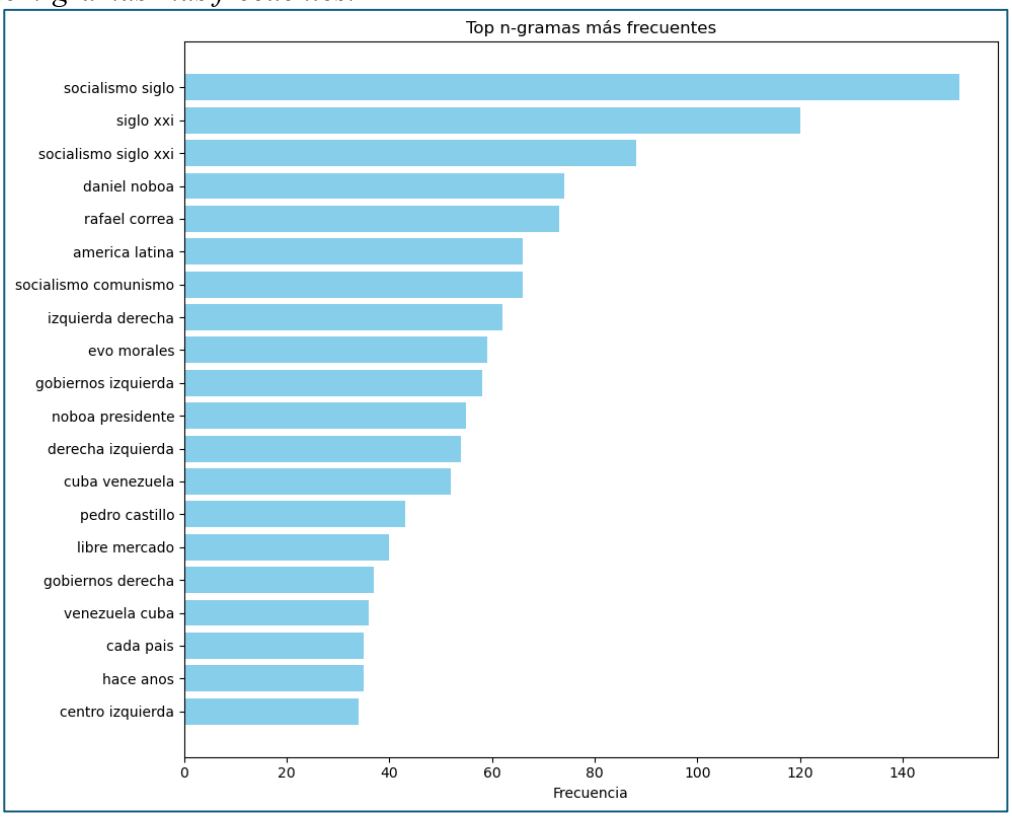
| ngrama | frecuencia |
|---|---|
| socialismo siglo | 151 |
| siglo xxi | 120 |
| socialismo siglo xxi | 88 |
| daniel noboa | 74 |
| rafael correa | 73 |
| america latina | 66 |
| socialismo comunismo | 66 |
| izquierda derecha | 62 |
| evo morales | 59 |
| gobiernos izquierda | 58 |
| noboa presidente | 55 |
| derecha izquierda | 54 |
| cuba venezuela | 52 |
| pedro castillo | 43 |
| libre mercado | 40 |
| gobiernos derecha | 37 |
| venezuela cuba | 36 |
| hace anos | 35 |
| cada pais | 35 |
| centro izquierda | 34 |

**N-Gramas más frecuentes**

En la figura 35 se observó la repetición de estos n-gramas en orden descendente, aquí se evidencia que las frases están direccionadas hacia el tema principal, posturas políticas, países latinos y algunas figuras políticas intervinientes del dentro del socialismo.

***Figura 36***

*Gráfico de n-gramas más frecuentes.*




**Clasificación de N-gramas en POS, NEG y NEU:**

Para un mayor análisis en los n-gramas se extrajeron todos estos para usar el modelo de inteligencia artificial más conocido, ChatGpt, para clasificar estos N-gramas en buenos, neutros o malos. A través de un Promt enviado por medio de la API de OpenAI se pudo clasificar estos n-gramas de forma masiva.(Ver Anexo – Porcedimiento para llamar a ChatGpt por API).

A continuación, se muestra el resultado de la clasificación por n-gramas:

Total de N-gramas: 50

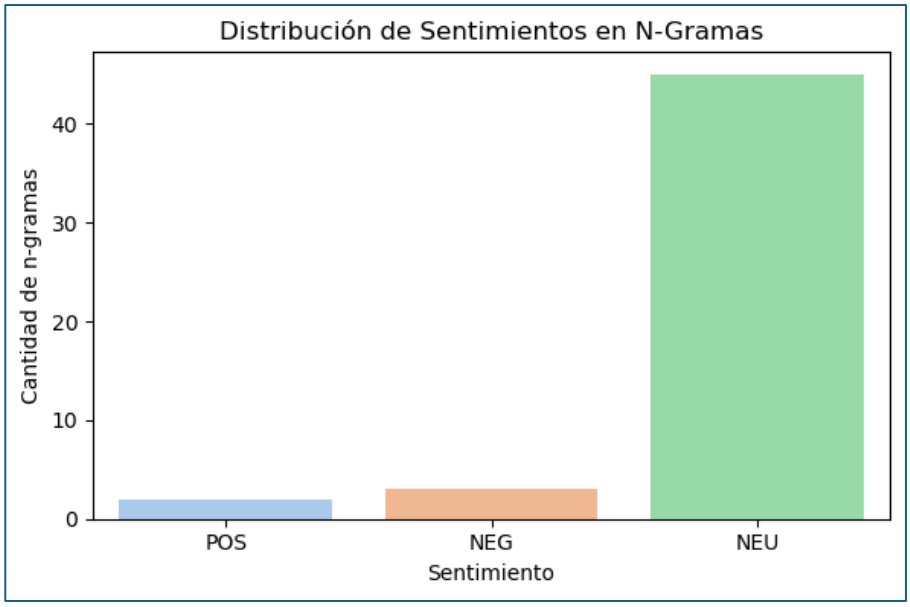
***Tabla 3***

*Tabla de distribución de N-gramas por clase*

| ***Sentimiento*** | ***Cantidad*** |
|---|---|
| ***POSITIVO*** | 2 |
| ***NEGATIVO*** | 3 |
| ***NEUTRO*** | 45 |

***Figura 37***

*Distribución de N-gramas por sentimiento*




El resultado mostró que la gran mayoría de frases fueron etiquetadas con la clase neutra y hubo un gran desbalance en los datos. Esto ayudo a ver que, si bien el tema es muy controversial y polarizado, el uso de bigramas o trigramas no es de gran utilidad en estos casos ya que no se muestra emociones explicitas.

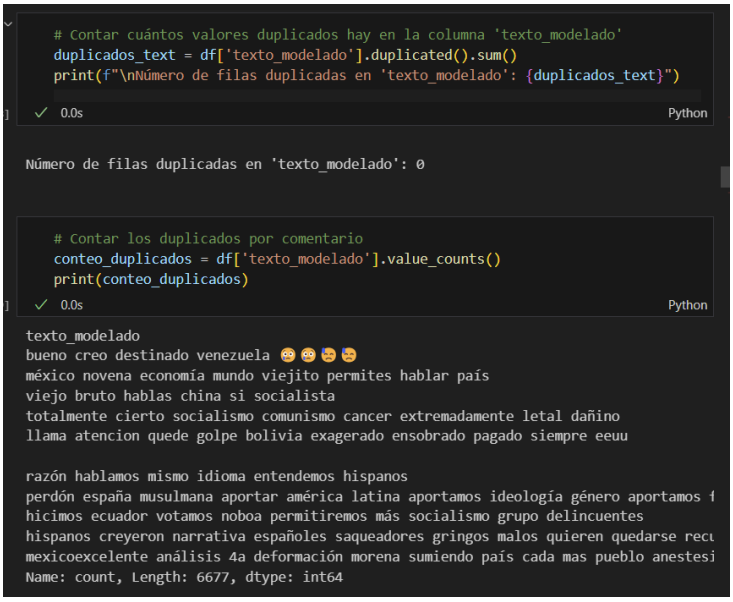
***Integrar los datos***

**Etiquetado de datos**:

Para etiquetar los datos se entrenó el modelo, con Emojis y StopWords relevantes a diferencia de la etapa de visualización. Los comentarios con esta limpieza se guardaron en el campo texto_modelado.

Se volvió a revisar que no haya comentarios repetidos después de la limpieza para etiquetar los comentarios.

***Figura 38***
*Verificación de posibles comentarios repetidos en "texto_modelado"*




Se puede observar que no hay valores duplicados por comentarios a pesar de mantener los StopWords y los emojis.

Se utilizo 3 formas distintas de etiquetado de los datos en modelos de análisis de sentimientos para tener un buen etiquetado en los datos, las estrategias fueron las siguientes:

**1.- Modelado con bert-base-multilingual-uncased-sentiment:**

- Este modelo está basado en BERT y es útil en el análisis de textos de varios idiomas, califica en una escala de 1 a 5.

**2.- Etiquetado con beto-sentiment-analysis**

- Se uso un modelo entrenado con el idioma español específicamente para entender el sentimiento explicito o implícito en los comentarios.

**3.- ChatGpt. 3.5-turbo.**

- Para una validación final, el modelo de ChatGpt también ayudo en la clasificación únicamente en los casos en los que los anteriores modelos disertaban en sus resultados.

**Modelo 1: Hugging Face**

Para implementar esta técnica, se usó la librería de transformes de Hugging Face. La clasificación de esta técnica asigna una apuntación del 1 al 5, lo cual se homologó de la siguiente manera (Ver Anexo - función clasificación de comentarios bert).

- 1, 2 estrellas = Sentimiento **NEG**
- 3 estrellas = Sentimiento **NEU**
- 4, 5 estrellas = Sentimiento **POS**

**Duración:** 7 minutos y 37,2 segundos

**Frecuencia por categoría:**

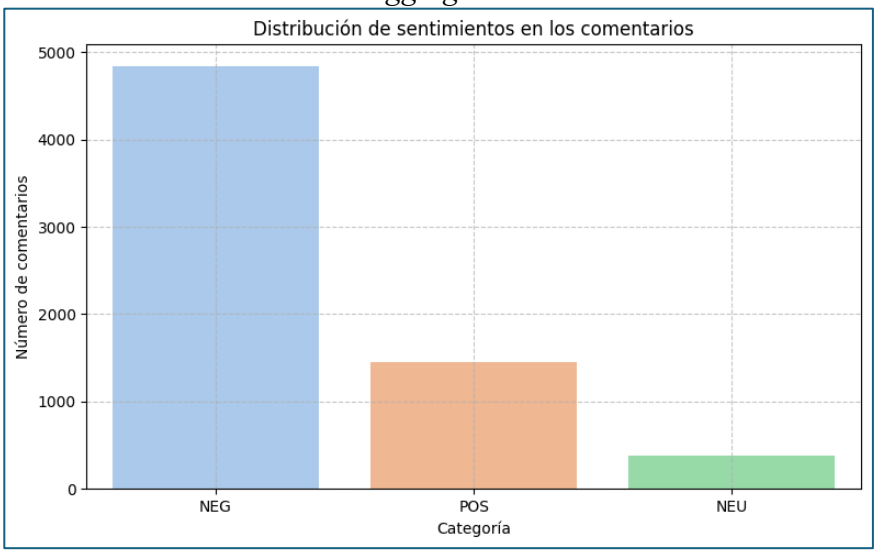
***Tabla 4***

*Frecuencia de sentimientos por modelo Hugging Face*

| *Sentimiento* | *Cantidad* |
|---|---|
| ***Positivo*** | ***1446*** |
| ***Negativo*** | ***4846*** |
| ***Neutro*** | ***384*** |

***Figura 39***

*Distribución de sentimientos con modelo Hugging Face*




**Modelo 2: Beto Sentiment Anlaysis**

Para mejorar la precisión en la clasificación de los comentarios y reducir error entre los modelos, se entrenó nuevamente los datos ahora con la técnica del modelo de beto sentiment análisis, El cual fue entrenado para clasificar textos en español, lo cual fue de gran utilidad para el contexto del tema principal. Este modelo ayudó a ver de mejor manera términos de contextualización en el idioma español. (Ver Anexo – Función Beto Sentiment Analysis)

**Duración:** 7 minutos 26,7 segundos.

**Frecuencia por categoría:**

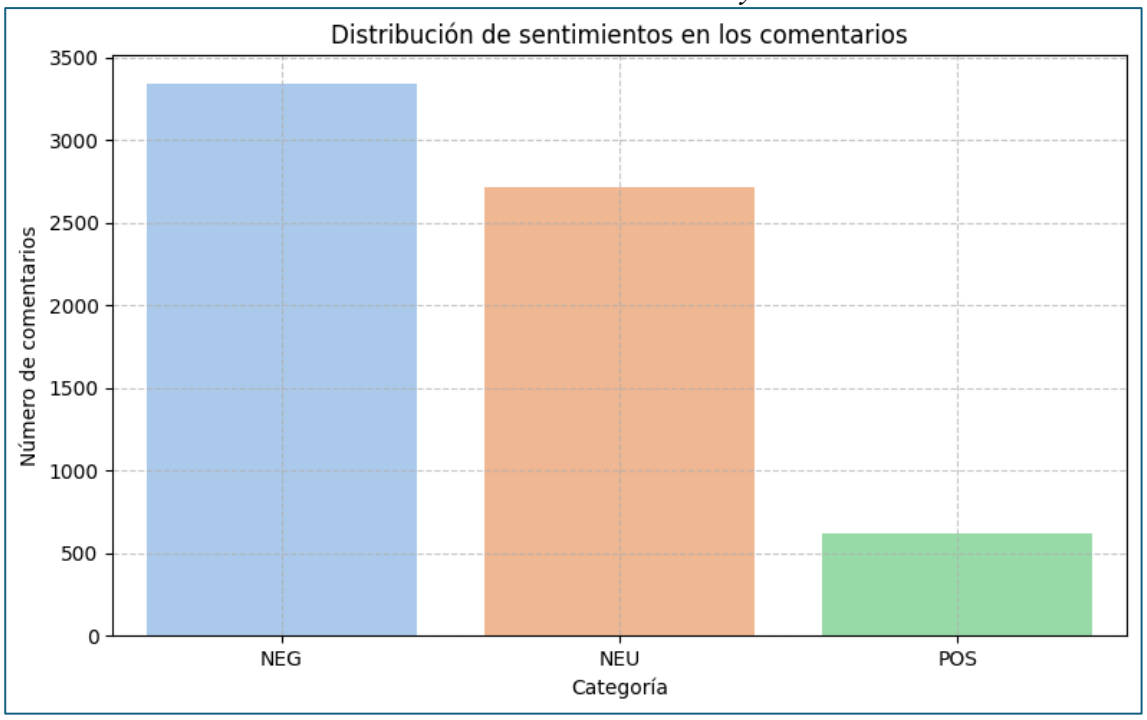
***Tabla 5***

*Frecuencia de sentimientos por modelo Beto Sentiment Analysis*

| *Sentimiento* | *Cantidad* |
|---|---|
| *Positivo* | *615* |
| *Negativo* | *3344* |
| *Neutro* | *2717* |

***Figura 40***

*Distribución de sentimientos con modelo Beto Sentiment Anlaysis*



**Modelo 3 ChatGpt:**

Estas fueron las diferencias que se obtuvo entre los dos modelos previamente clasificados.

***Figura 41***

Diferencias en resultados entre modelos Hugging Face y beto-sentiment-analysis

| | texto_modelado | sentimiento | sentimiento_esp |
|---|---|---|---|
| 0 | méxico novena economía mundo viejito permites ... | POS | NEU |
| 6 | hispanos creyeron narrativa españoles saqueado... | NEG | NEU |
| 8 | perdón españa musulmana aportar américa latina... | NEG | NEU |
| 9 | razón hablamos mismo idioma entendemos hispanos | NEG | NEU |
| 10 | peru socialismo habra | NEU | NEG |
| 12 | lis tres paises mayoria indigena😱😱 | POS | NEU |
| 17 | sirve asgueriso sosialismo siglo 20 | POS | NEU |
| 23 | ojalá peruanos ciegos engañados darse cuenta d... | NEG | NEU |
| 24 | chile | POS | NEU |
| 33 | exelente programaes verdadera visionde politic... | POS | NEU |

Se guardo en DataSets los registros con resultados diferentes e iguales para realizar etiquetado de comentarios con diferente etiquetado con la API de ChatGPT.

***Figura 42***

*Guardar Diferencias y resultados iguales.*

```
#Se guarda los comentarios con diferencias
diferencias.to_csv("diferencias.csv", index=False, encoding="utf-8")


iguales = df[df['sentimiento'] == df['sentimiento_esp']]
iguales.to_csv("iguales.csv", index=False, encoding="utf-8")
✓ 0.1s
```

Se ejecuto con la APIkey de OPEN AI para clasificar los comentarios con diferencias.

**Duración:** 27minutos con 13.4 segundos.

***Figura 43***

*Etiquetado con ChatGpt con resultados distintos de modelos de clasificación*

```
Clasificando comentarios con ChatGPT: 100%|██████████| 2973/2973 [27:23<00:00,  1.8
Clasificación completada y guardada en 'comentarios_clasificados_chatgpt.csv'!

C:\Users\Usuario\AppData\Local\Temp\ipykernel_12800\1395456628.py:11: SettingWithCop
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
  diferencias['sentimiento_final'] = diferencias['texto_modelado'].astype(str).progr
```

Después de haber finalizado con el etiquetado de los datos con diferencias, se consolido con los datos restantes para ver la nueva distribución por sentimientos. Lo que se hizo fue tomar el CSV donde la clasificación coincidía en los 2 primeros modelos y unir los comentarios restantes que fueron clasificados por ChatGpt.

***Figura 44***

*Proceso integrar los comentarios*

```
[67]  #Filtramos las columnas que vamos a necesitar
      diferencias = diferencias[['text', 'texto_limpio', 'sentimiento_final']]
      ✓ 0.0s

[69]  iguales = pd.read_csv("iguales.csv")
      ✓ 0.0s

[70]  #Filtramos las columnas que vamos a necesitar
      iguales = iguales[['text', 'texto_limpio', 'sentimiento']]
      ✓ 0.0s

[71]  iguales = iguales.rename(columns={'sentimiento': 'sentimiento_final'})
      ✓ 0.0s

[75]  iguales.columns
      ✓ 0.0s
...   Index(['text', 'texto_limpio', 'sentimiento_final'], dtype='object')

[76]  diferencias.columns
      ✓ 0.0s
...   Index(['text', 'texto_limpio', 'sentimiento_final'], dtype='object')
```

**DataSet DF_FINAL**

Después de haber consolidado e integrado la información, a continuación de ello, se muestra la distribución del DataSet final.

**Frecuencia por categoría**

***Tabla 6***

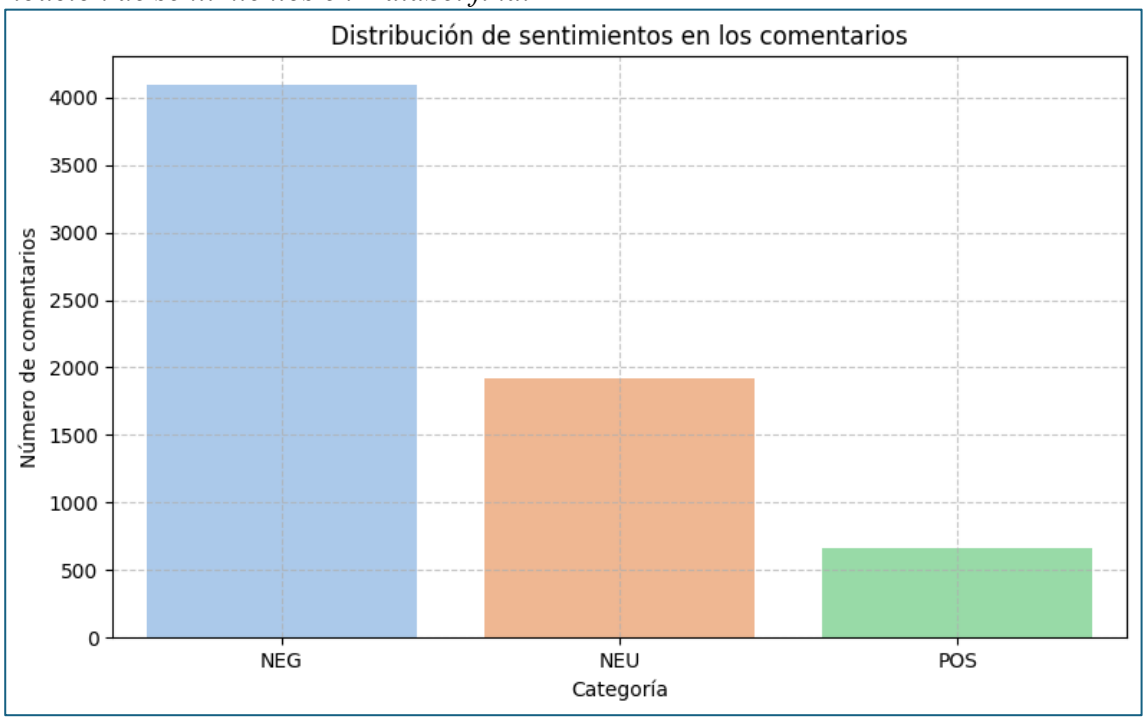
*Frecuencia de sentimientos DataSet final*

| *Sentimiento* | *Cantidad* |
|---|---|
| *Positivo* | *663* |
| *Negativo* | *4098* |
| *Neutro* | *1915* |

**Gráfico de distribución por sentimientos:**

***Figura 45***

*Distribución de sentimientos en DataSet final*




Como se pudo observar, hubo un desbalance y una prioridad hacia la clase NEG, por lo que trabajar con estos datos, puede ocasionar poca precisión en los modelos, para ello, Se realizo un submuestreo y un sobremuestreo para trabajar con la misma cantidad de registros para las tres categorías. Se uso técnicas de balanceo para tener la misma cantidad de registros. La base fue la clase NEU, donde el resto de las clases se igualaron a su frecuencia de registros. Para la clase NEG únicamente se eliminó los comentarios de forma randomica para igualar a la clase NEU, mientras que para la clase POS, se los duplico randomicamente. A continuación, se muestra el resultado de este balanceo. Este balanceo de carga se guardó en un DataSet llamado “df_submuestro”.

**Frecuencia por categoría:**

***Tabla 7***

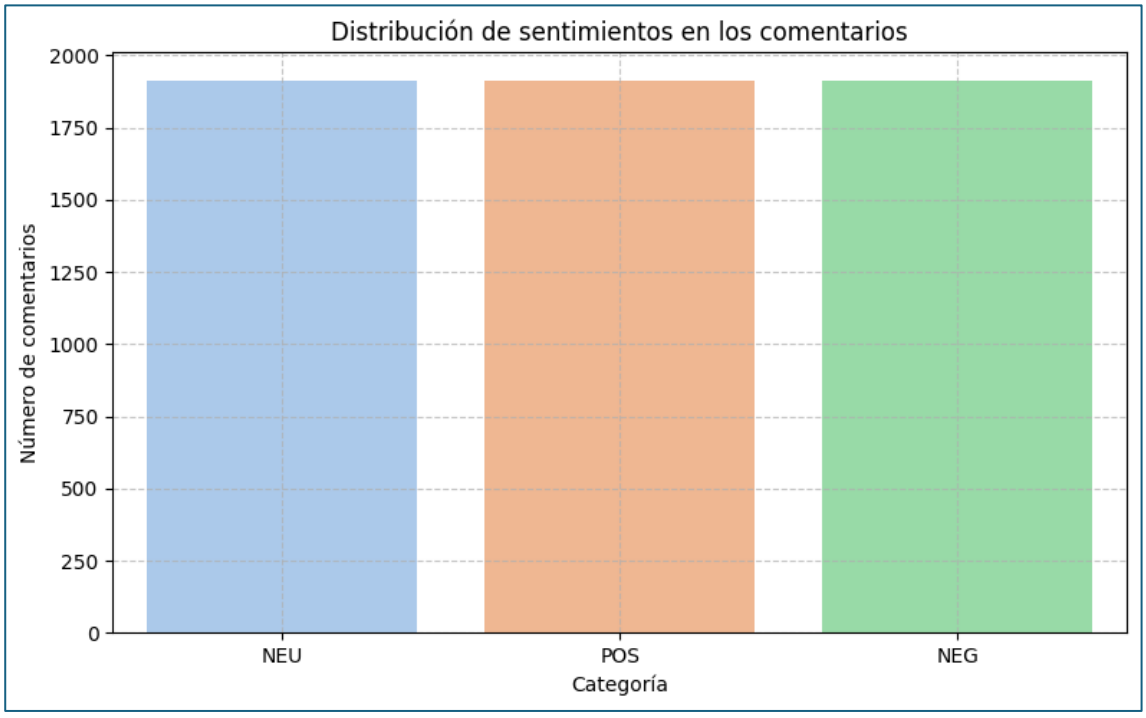
*Frecuencia de sentimientos DataSet con submuestreo*

| *Sentimiento* | *Cantidad* |
|---|---|
| *Positivo* | *1915* |
| *Negativo* | *1915* |
| *Neutro* | *1915* |

**Gráfico de distribución por sentimientos:**

***Figura 46***

*Distribución de sentimientos DataSet con Submuestreo*



***Formatear los datos***

Antes de entrenar los modelos, fue necesario codificar las clases en valores numéricos para entrenar los algoritmos de aprendizaje automático.

Se usó LabelEncoder de la librería Scikit-learn para clasificar los comentarios en positivo negativo y neutro tanto para el DataSet final y el DataSet con submuestreo, usando la siguiente codificación:

***Figura 47***

*Tabla con valores de codificación por sentimiento*

| ***Sentimiento*** | ***Código*** |
|---|---|
| ***POS*** | 2 |
| ***NEU*** | 1 |
| ***NEG*** | 0 |

***Figura 48***

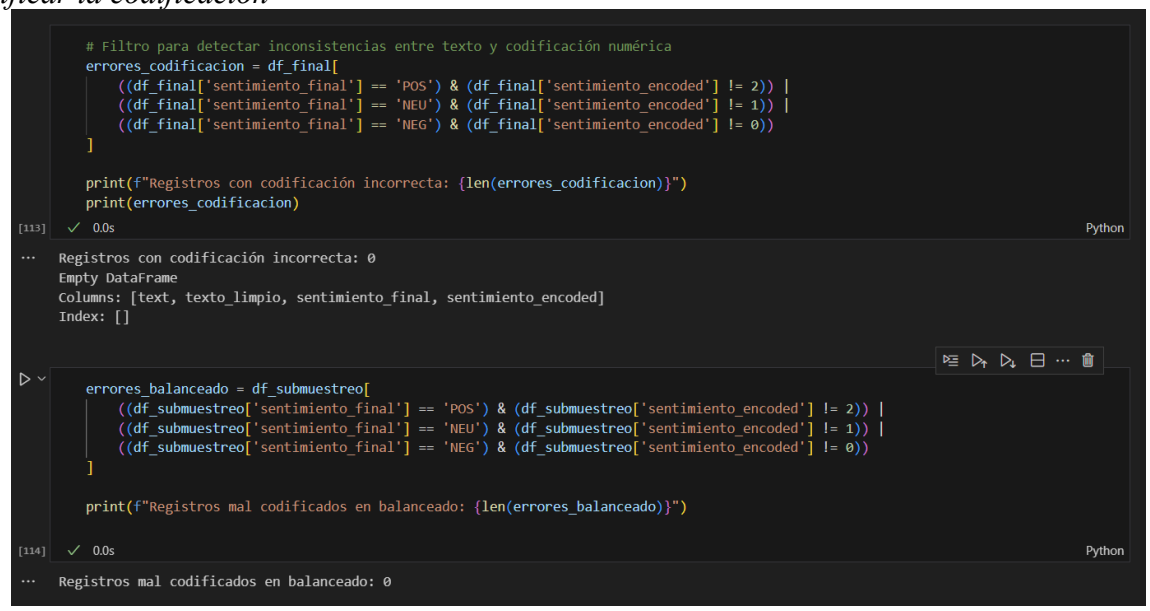
*Codificación con LabelEncoder*


**Validación de codificación.**

Se realizo una validación de la codificación para cuando el valor de sentimiento sea diferente de su valor correspondiente codificado. Se uso condiciones lógicas para comprar ambas columnas la que tenía la clasificación en texto y el otro campo con la clasificación codificada.

***Figura 49***

*Verificar la codificación*




## FASE 4: Modelado

### *Seleccionar técnicas de modelado*

Para realizar el modelado de sentimientos sobre los comentarios previamente seleccionados y preparados, se deicidio aplicar una combinación de modelos tradicionales de aprendizaje automático y modelos de lenguaje de gran escala (LLMs).

Se escogieron cinco algoritmos: tres tradicionales y dos modelos basados en inteligencia artificial comúnmente usados hoy en día.

Cada modelo se entrenó con dos DataSets diferentes:

- Con el conjunto de datos original **(df_final)**.
- Con el conjunto balanceado realizado con submuestreo **(df_submuestreo)**.

Al final se obtuvo un total de 10 modelos. Además de esos, se aplicó una mejora con hiperparámetros, validación cruzada y GridSearch únicamente en los modelos tradicionales.

Los modelos seleccionados son los siguientes:

**1. Multinomial Naive Bayes (MNB):**

Este algoritmo si es de utilidad en clasificación de texto gracias a que es eficiente y su implementación no es muy compleja. Su función se basa en la probabilidad de que una palabra aparezca en un documento el cual pertenece a una clase determinada. Funciona mejor cuando se usa vectores de características TF-IDF o conteo de palabras. También es bueno con texto de alta dimensión y también al momento de entrenar y predecir los datos son rápidos.

**2.- Random Forest**

Random forest construye varios árboles de decisión y agrega sus predicciones. Ayudó considerablemente ya que es bien robusto con el sobreajuste y controla bien los datos mixtos o con ruido, datos que están bien presentes en comentarios de redes sociales como YouTube. también se destaca por manejar bien las relaciones no lineales entre variables y tiene una alta precisión y tolera bien con características en los datos irrelevantes o redundantes.

**3.- Support Vector Machines (SVM)**

Las máquinas de vectores de soporte para clasificación. son eficaces con en espacios de alta dimensión y con conjuntos de datos que son pequeños, lo cual se tiene en ambos DataSets, tienen un buen desempeño con textos representados como vectores, ideal para clasificaciones multiclase, sin embargo, puede ser un poco malo con datos desbalanceados.

**5.- XGBoost**

Se escogió para su análisis debido a su buen rendimiento en tareas de clasificación, específicamente en contextos donde hay ruido en los datos o hay características redundantes o estructuras un poco complejas. Al igual que Random Forest usa arboles de decisión optimizados, pero usando técnicas de gradiente y regularización.

**5.- ChatGPT**

ChatGpt es un modelo LLM (Large Language Model) que ya es preentrenado con una enorme cantidad de texto y fine tuneado para comprender el lenguaje natural, lo cual es super eficiente para comentarios en redes sociales. Puede entender el lenguaje complejo, varios idiomas, ironía, el contexto sociopolítico y sarcasmo. Es excelente para analizar el contexto sociopolítico implícito.

**6.- DeepSeek**

DeepSeek, al igual que el modelo anterior, es un LLM de alto rendimiento, orientado a comprender cualquier tipo de texto, es útil para analizar el texto corto, como los comentarios del DataSet. Últimamente, ha sido competencia con ChatGPT lo que hace de gran ayuda para analizar cuál es mejor modelo.

***Generar diseño de prueba***

Se procedió a dividir los datos en conjuntos de entrenamiento y prueba en los datos para df_final y df_submuestreo.

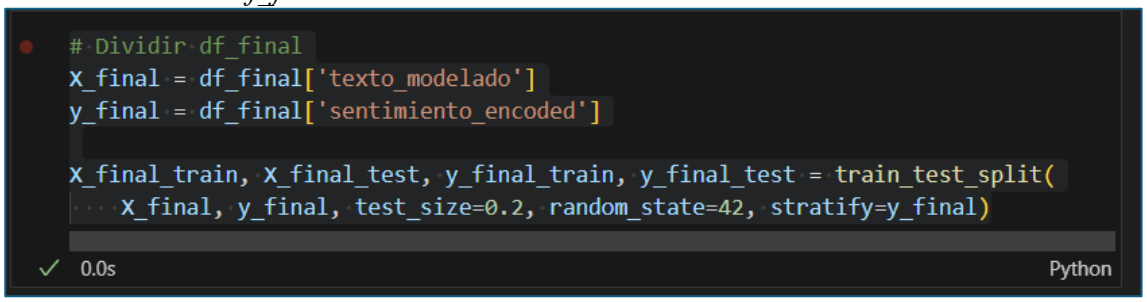
Se utilizo la función train_test_split de scikit-learn con las siguientes características.

- Test_size = 0.2: el 20% de los datos para pruebas el 80% para entrenamiento.
- Random_state = 42: Se estableció una semilla fija para próximas ejecuciones.
- Stratify = y: Mantener la proporción de clases en ambos datos (train y test).

**df_final:**

***Figura 50***

*Preparación de datos df_final*




**Distribución de datos en entrenamiento y prueba:**

**Entrenamiento:**

***Tabla 8***

*Distribución de datos de entrenamiento df_final*

| *Sentimiento* | *Codificación* | *Cantidad* |
|---|---|---|
| *POSITIVO* | *2* | *530* |
| *NEGATIVO* | *0* | *3278* |
| *NEUTRO* | *1* | *1532* |

**Prueba:**

***Tabla 9***

*Distribución de datos de prueba df_final*

| *Sentimiento* | *Codificación* | *Cantidad* |
|---|---|---|
| *POSITIVO* | *2* | *133* |
| *NEGATIVO* | *0* | *820* |
| *NEUTRO* | *1* | *383* |

**df_submuestreo:**

***Figura 51***

*Preparación de los datos df_submuestreo.*

```
# Dividir df_submuestreo
X_sub = df_submuestreo['texto_modelado']
y_sub = df_submuestreo['sentimiento_encoded']

X_sub_train, X_sub_test, y_sub_train, y_sub_test = train_test_split(
    X_sub, y_sub, test_size=0.2, random_state=42, stratify=y_sub)
```

✓ 0.0s Python

**Distribución de datos en entrenamiento y prueba:**

**Entrenamiento:**

***Tabla 10***

*Distribución de datos de entrenamiento df_submuestreo*

| *Sentimiento* | *Codificación* | *Cantidad* |
|---|---|---|
| *POSITIVO* | *2* | *1532* |
| *NEGATIVO* | *0* | *1532* |
| *NEUTRO* | *1* | *1532* |

**Prueba:**

***Tabla 11***

*Distribución de datos de prueba df_submuestreo*

| *Sentimiento* | *Codificación* | *Cantidad* |
|---|---|---|
| *POSITIVO* | *2* | *383* |
| *NEGATIVO* | *0* | *383* |
| *NEUTRO* | *1* | *383* |

***Construcción del modelo***

1. **Algoritmo: MultinomialNB**

- ***Entrenamiento con DataSet df_final***

Para el entrenamiento de Naive bayes, se implementó un flujo con un Pipeline, el cual manejo dos elementos muy importantes, "TfidfVectorizer" para vectorizar el texto en numérico en base a la frecuencia relativa de cada término, para así poder minimizar el impacto de las palabras más repetitivas o comunes y dar protagonismo a las palabras que son más representativas. También se usó MultinomialNB, clasificador probabilístico, el cual es útil en comentarios discretos o los que son generados con conteo de palabras o TF-IDF. (Ver Anexo – Modelo Naive Bayes df_final)

**Evaluación del modelo Naive Bayes:**

***Tabla 12***
*Evaluación modelo Naive Bayes df_final*

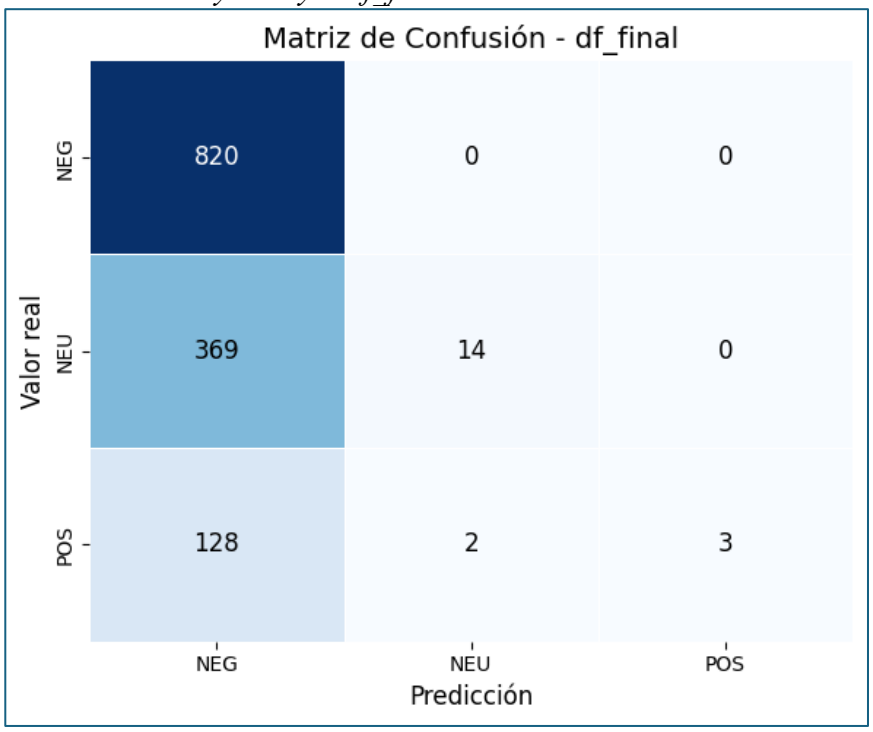
| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.62 | 1.00 | 0.77 | 820 |
| ***NEU*** | 0.88 | 0.04 | 0.07 | 383 |
| ***POS*** | 1.00 | 0.02 | 0.04 | 133 |
| ***Accuracy*** | | | 0.63 | 1336 |

| | | | | |
|---|---|---|---|---|
| ***Macro avg*** | 0.83 | 0.35 | 0.29 | 1336 |
| ***Weighted avg*** | 0.73 | 0.63 | 0.50 | 1336 |

**Matriz de confusión:**

***Figura 52***

*Matriz de confusión modelo Naybe bayes df_final*




Después del entrenamiento, se observó que el modelo clasifico todo casi todo en clase NEG y se evidenció una mala clasificación en las clases POS y NEU. Esto ocurrió debido al desbalanceo de los datos.

***Implementación de mejora al modelo df_final***

A partir de los resultados anteriores, se realizó un mejorado en el modelado, se utilizó la técnica de optimización de hiperparámetros y por GridSearch combinado con validación cruzado. Estas técnicas ayudaron a tener un mejor rendimiento para el modelo, una mejor configuración para el vectorizador TF-IDF (Ver Anexo – Modelo mejorado NaiveBayes df_final). Los hiperparámetros que se ajustaron fueron:

- **Ngram_range**: Se uso unigramas (1,1) y bigramas (1,2).
- **Min_df:** Se evaluó valores de corte con frecuencia mínima en las palabras: 1,2,5.
- **Max_df:** Se evaluó límetes superiores para no tomar en cuenta palabras muy frecuentes: 0.9, 0.95, 1.0.
- **Alpha:** Se suavizo para el clasificador: 0.1, 0.5, 1.0.

**Evaluación del modelo Naive Bayes:**

***Tabla 13***
*Evaluación modelo Naive Bayes df_final mejorado*

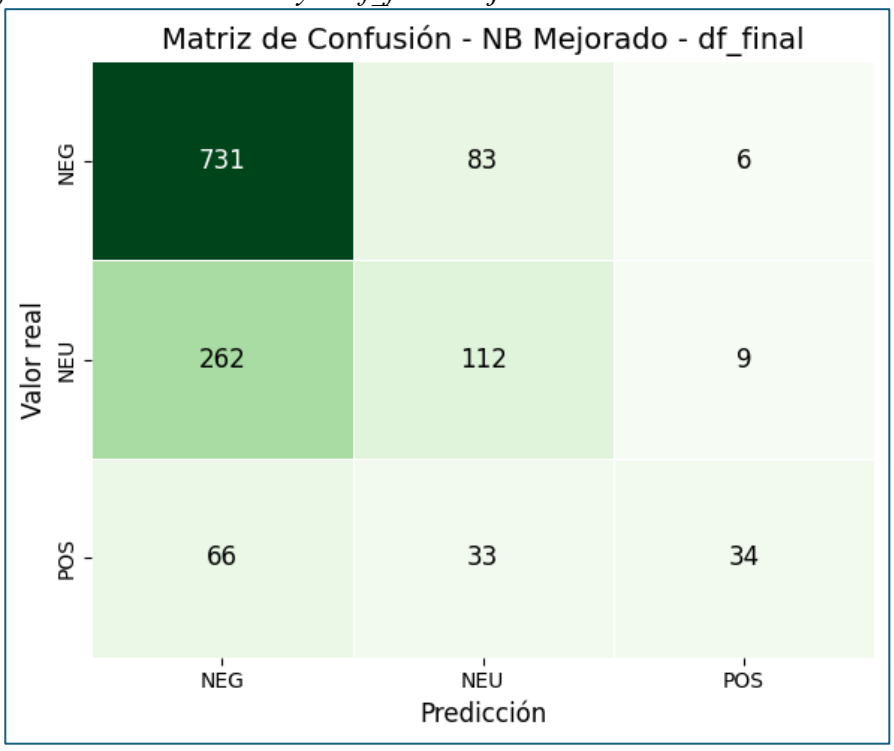
| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.69 | 0.89 | 0.78 | 820 |
| ***NEU*** | 0.49 | 0.29 | 0.37 | 383 |
| ***POS*** | 0.69 | 0.26 | 0.37 | 133 |
| ***Accuracy*** | | | 0.66 | 1336 |

| | | | | |
|---|---|---|---|---|
| ***Macro avg*** | 0.63 | 0.48 | 0.51 | 1336 |
| ***Weighted avg*** | 0.63 | 0.66 | 0.62 | 1336 |

**Matriz de confusión:**

***Figura 53***

*Matriz de confusión modelo NaiveBayes df_final mejorado*




Gracias a el ajuste de estos hiperparámetros se obtuvo una mejora estable a diferencia del modelo anterior, hubo una mejora en las clases NEU y POS.

- ***Entrenamiento con DataSet df_submuestreo***

Se entrenó el modelo de NaiveBayes con el DataSet balanceado (df_submuestreo). Este entrenamiento también se usó Pipeline, donde se usó TfidfVectorizer y el clasificador MultinomialNB. Para un mejor entrenamiento se usó en esta primera instancia GridSearchCV ajustando los siguientes hiperparámetros (Ver Anexo – Modelado NaiveBayes df_submuestreo mejorado):

- **ngram_range:** (1,1) en unigramas y (1,2) para bigramas.
- **min_df:** 1,2,5 para no tomar en cuenta términos con baja frecuencia.
- **max_df:** 0.9, 0.95, 1.0 para no tomar en cuenta términos de alta frecuencia.
- **Alpha:** 0.1, 0.5, 1.0 suavizamiento de Laplace.
- **Cv:** 5 particiones

**Evaluación del modelo Naive Bayes:**

***Tabla 14***

*Evaluación modelo NaybeBayes df_submuestreo*

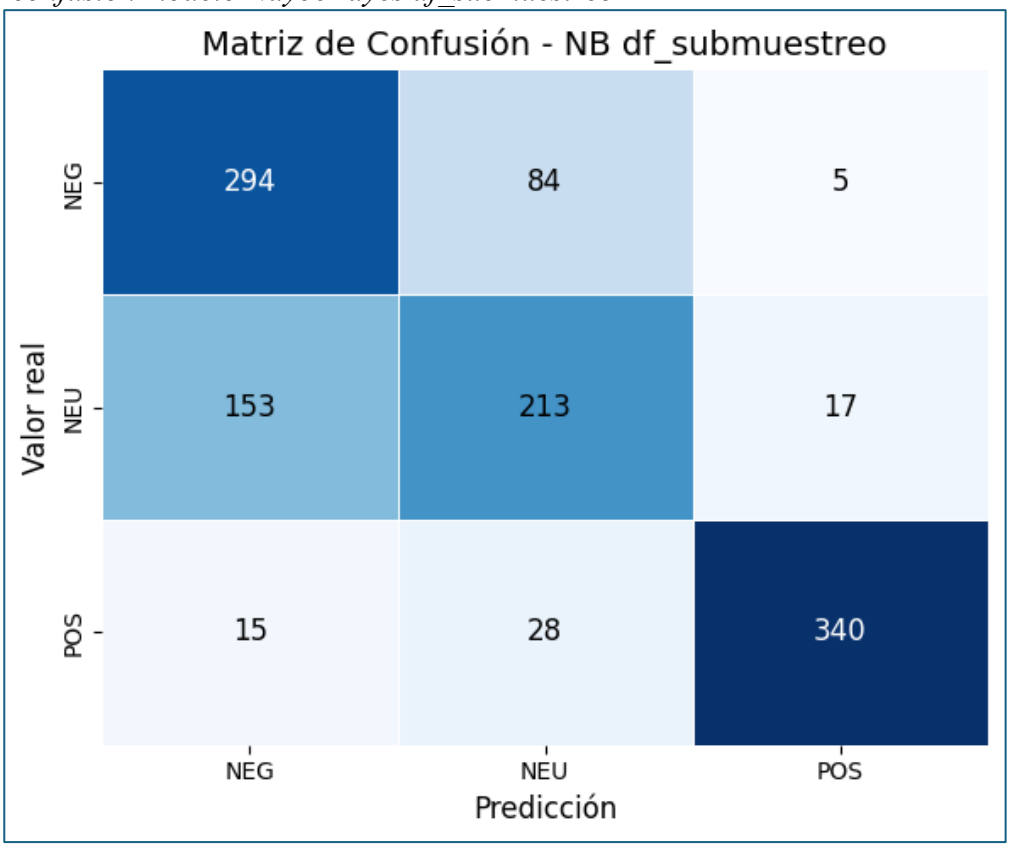
| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.64 | 0.77 | 0.70 | 383 |
| ***NEU*** | 0.66 | 0.56 | 0.60 | 383 |
| ***POS*** | 0.94 | 0.89 | 0.91 | 383 |
| ***Accuracy*** | | | 0.74 | 1149 |

| | | | | |
|---|---|---|---|---|
| ***Macro avg*** | 0.74 | 0.74 | 0.74 | 1149 |
| ***Weighted avg*** | 0.74 | 0.74 | 0.74 | 1149 |

**Matriz de confusión**

***Figura 54***

*Matriz de confusión modelo NaybeBayes df_submuestreo*



Después del entrenamiento se pudo notar una gran mejora y se observó que ahora las clases son más homogéneas entre clases, Hubo mejor capacidad para predecir los comentarios en clases POS y NEU. El modelo fue más equilibrado entre las 3 clases.

***Implementación de mejora al modelo df_submuestreo***

Se trató de mejorar el modelo anterior, se usó CountVectorizer, esta técnica tuvo un mejor desempeño, para este proceso se usaron los siguientes parámetros (Ver Anexo – Modelado NaiveBayes df_submuestreo mejorado):

- **vect__ngram_range:** (1,1) en unigramas y (1,2) para bigramas.
- **vect__min_df:** 1 y 2 frecuencia mínima.
- **vect__max_df:** 0.9 y 0,95 para ignorar términos con mucha frecuencia.
- **nb__alpha:** 0.1, 0.5 y 1.0 para el suavizado Laplace.

**Evaluación del modelo Naive Bayes:**

***Tabla 15***
*Evaluación NaiveBayes mejorado df_submuestreo*

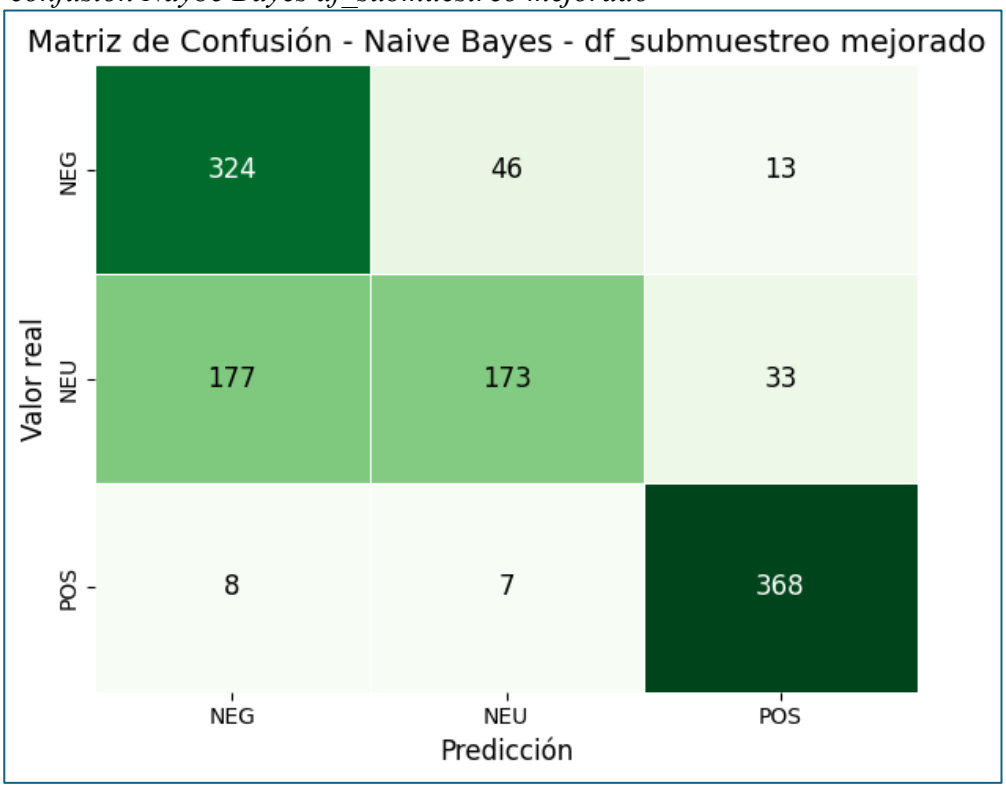
| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.64 | 0.85 | 0.73 | 383 |
| ***NEU*** | 0.77 | 0.45 | 0.57 | 383 |
| ***POS*** | 0.89 | 0.96 | 0.92 | 383 |

| | | | | |
|---|---|---|---|---|
| ***Accuracy*** | | | 0.75 | 1149 |
| ***Macro avg*** | 0.76 | 0.75 | 0.74 | 1149 |
| ***Weighted avg*** | 0.76 | 0.75 | 0.74 | 1149 |

**Matriz de confusión:**

***Figura 55***

*Matriz de confusión Naybe Bayes df_submuestreo mejorado*



Este mejorado fue el mejor considerablemente, debido a que la clase POS tuvo un rendimiento bueno, la clase NEU tuvo una alta precisión, pero a veces se confundía con la clase NEG.

***Resumen de evaluación algoritmo Naive Bayes.***

- **1 modelo: Naive Bayes - df_final (sin mejoras)**
    - Clasifica casi todo como NEG.
    - NEU y POS tienen un bajo recall.
    - Alto sesgo en la clase NEG.
- **2 modelo: Naive Bayes - df_final (mejorado)**
    - Detecta NEU y POS, pero con un grado de dificultad.
    - Subió un poco ligeramente.
    - Más balanceado los resultados, pero igual hay dificultad con las clases NEU y POS.
- **3 modelo: Naive Bayes - df_submuestreo (sin mejora)**
    - Más equilibrado
    - Clasifica bien la clase positiva **(POS)**.
    - Buen rendimiento (F1 promedio. 0.74).
- **4 modelo: Naive Bayes - df_submuestreo (mejorado)**
    - Buena precisión en POS.
    - El Recall mejoro para NEU.
    - El Accuracy y macro tuvo un resultado más alto.

***Tabla de comparación modelos Naive Bayes:***

***Tabla 16***

*Comparación de modelos Naive Bayes.*

| *Modelo* | *Accuracy* | *F1 Macro* | *Bueno* |
|---|---|---|---|
| *NB - df_final* | 0.63 | 0.29 | NEG (únicamente) |
| *NB Mejorado - df_final* | 0.66 | 0.51 | NEG, ligeramente NEU/POS |
| *NB - df_submuestreo* | 0.74 | 0.74 | POS, balanceado en general |
| *NB - df_submuestreo mejorado* | 0.75 | 0.74 | Mejor resultado global |

**Conclusión General:**

El mejor modelo fue el entrenado con df_submuestreo mejorado debido a que mejoró el F1 macro y accuracy, hay un mejor balance entre las clases POS, NEU y NEG. También, reconoció un poco mejor las clases POS y NEU sin perder precisión en la clase NEG.

## 2. Algoritmo: Random Forest

- ***Entrenamiento con DataSet df_final***

Se entrenó el modelo de Random Forest con el DataSet desbalanceado (df_final). Random forest es un buen modelo ya que puede manejar entrenamientos de multiclase, tiene un buen rendimiento en sobreajuste y trabaja bien con DataSets de varias características. (Ver Anexo - modelado Random Forest df_final)

Se uso técnicas de vectorización de texto con TF-IDF por medio de TfidfVectorizer, también se usó RandomForestClassifier de sklearn.ensemble. con una semilla de 42.

**Evaluación del modelo Random Forest:**

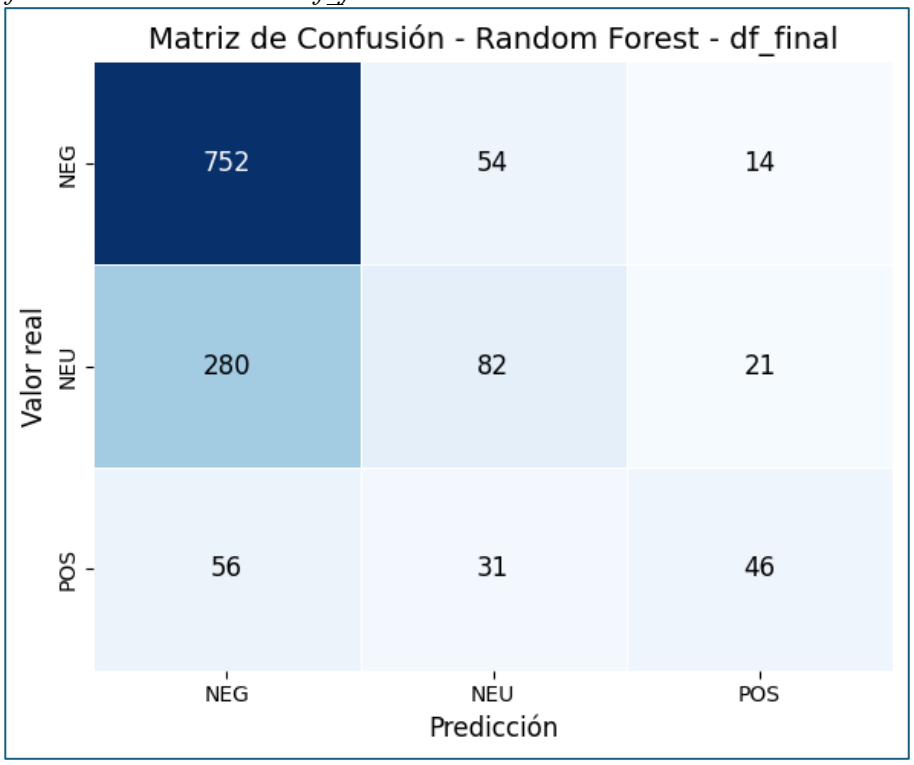
***Tabla 17***
*Evaluación Random Forest df_final*

| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.69 | 0.92 | 0.79 | 820 |
| ***NEU*** | 0.49 | 0.21 | 0.30 | 383 |
| ***POS*** | 0.57 | 0.35 | 0.43 | 133 |

| | | | | |
|---|---|---|---|---|
| ***Accuracy*** | | | 0.66 | 1336 |
| ***Macro avg*** | 0.58 | 0.49 | 0.51 | 1336 |
| ***Weighted avg*** | 0.62 | 0.66 | 0.61 | 1336 |
| ***Gini Index*** | | | | 0.5213 |

***Figura 56***

*Matriz de confusión Random Forest df_final*




El modelo tuvo un buen rendimiento en la clase NEG, pero tuvo inconvenientes en las clases NEU y POS, esto pudo ocurrir debido a que hay el desbalance entre clases. El índice

Gini obtuvo un 0.5213 que indicó que tiene una discriminación aceptable, por lo que si se debe considerar un ajuste para mejorar en las clases más débiles.

***Implementación de mejora al modelo df_final***

Para mejorar el modelo se realizó una optimización que incluyó vectorización y una búsqueda de los mejores hiperparámetros por medio de validación cruzada (Ver Anexo – Modelado Random Forest df_final. Mejorado).

- **ngram_range: (1, 2)** Unigramas y bigramas.
- **min_df =5 y max_df= 0.95:** Eliminar términos más frecuentes.
- **sublinear_tf:** True, Aplicar escalado logarítmico a las frecuencias.
- **n_estimators:** Número de muestras para dividir (100, 200).
- **max_depth:** 20 y no se definió un límite.
- **min_samples_split:** Número mínimo de muestras para dividir dentro de un nodo 2,5.
- **max_features:** Selección de características (sqrt, log2).
- **class_weight:** 'Balanced', mitigar el desbalance entre las clases.

**Evaluación del modelo Random Forest:**

***Tabla 18***

*Evaluación modelo Random Forest df_final mejorado*

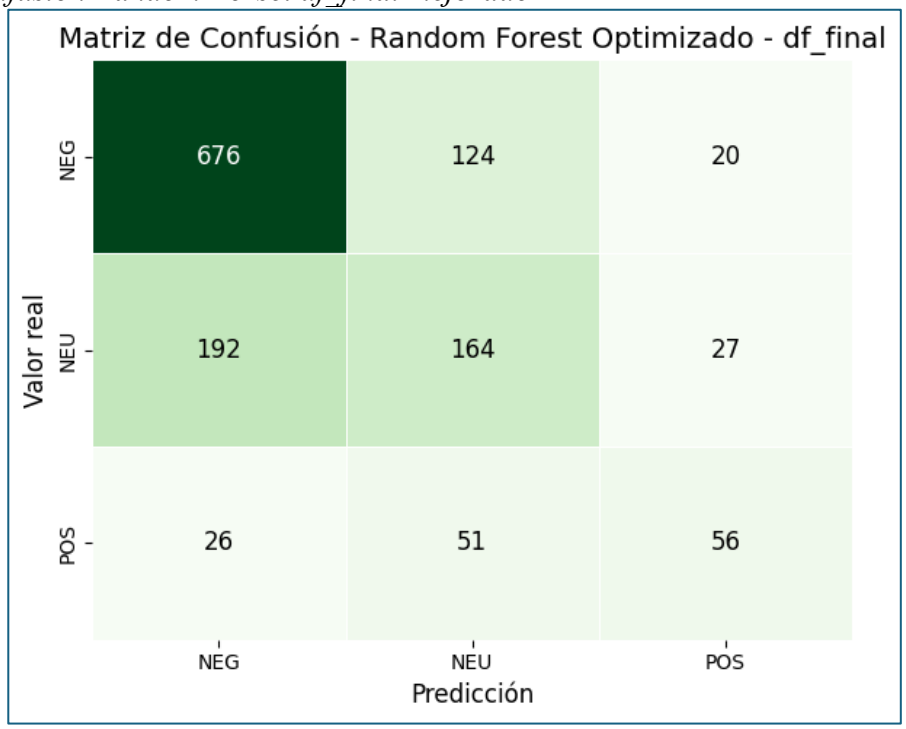
| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.76 | 0.82 | 0.79 | 820 |
| ***NEU*** | 0.48 | 0.43 | 0.45 | 383 |

| | | | | |
|---|---|---|---|---|
| ***POS*** | 0.54 | 0.42 | 0.47 | 133 |
| ***Accuracy*** | | | 0.67 | 1336 |
| ***Macro avg*** | 0.59 | 0.56 | 0.57 | 1336 |
| ***Weighted avg*** | 0.66 | 0.67 | 0.66 | 1336 |
| ***Gini Index*** | | | | 0.5484 |

**Matriz de confusión:**

***Figura 57***

*Matriz de confusión Random Forset df_final mejorado*



Este entrenamiento mejorado si funciono, La clase NEG, se mantuvo siendo la mejor, pero, por otro lado, las clases NEU y POS mejoraron debido a el vectorizador ajustado. Sin embargo, la precisión de clases no fue buenas del todo.

- ***Entrenamiento con DataSet df_submuestreo***

Para tener un mejor rendimiento se entrenó con el DataSet df_submuestreo, donde se buscó mejorar las categorías en NEU y POS (Ver Anexo – Modelado Random Forest df_submuestreo.)

- **ngram_range= (1, 2):** Capturar unigramas y bigramas
- **min_df = 5, max_df= 0.95:** Filtrar palabras raras o muy frecuentes.
- **sublinear_tf= True:** Se aplicó penalización logarítmica sobre la frecuencia.
- **n_estimators:** 100 y 200 árboles.
- **max_depth:** Profundidad máxima del árbol 20.
- **min_samples_split:** Número mínimo de muestras para dividir el nodo (2 o 5).
- **max_features:** Número máximo de características consideradas (sqrt o log2).
- **class_weight:** 'balanced' sirvió para equilibrar el peso de cada clase automáticamente.

**Evaluación del modelo Random Forest:**

***Tabla 19***

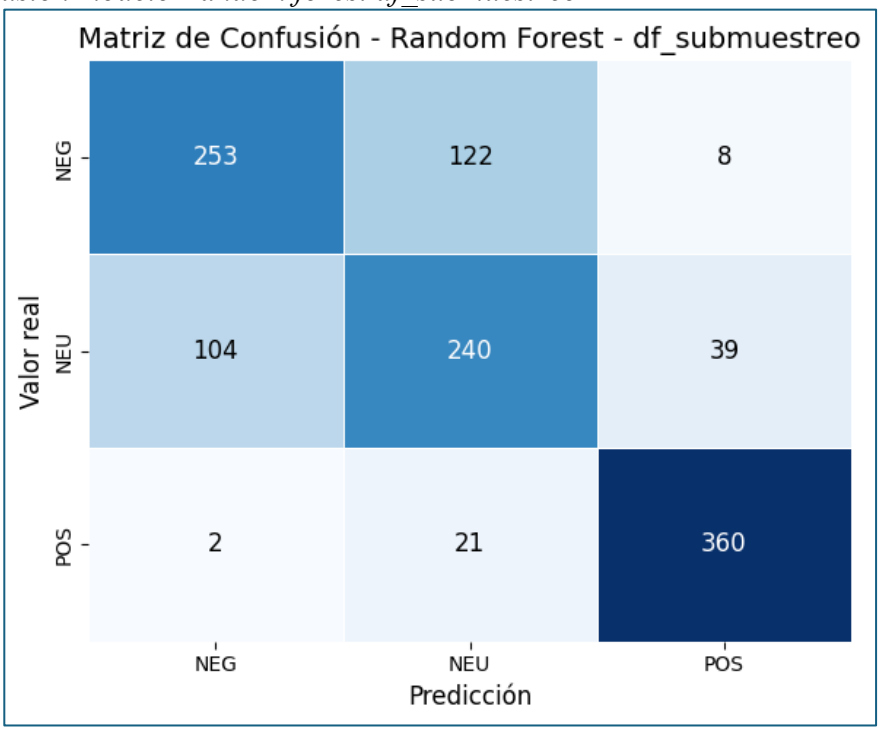
*Matriz de confusión modelo Random Forest df  submuestreo*

| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.70 | 0.66 | 0.68 | 383 |
| ***NEU*** | 0.63 | 0.63 | 0.63 | 383 |
| ***POS*** | 0.88 | 0.94 | 0.91 | 383 |
| ***Accuracy*** | | | 0.74 | 1149 |
| ***Macro avg*** | 0.74 | 0.74 | 0.74 | 1149 |
| ***Weighted avg*** | 0.74 | 0.74 | 0.74 | 1149 |
| ***Gini Index*** | | | | 0.7772 |

**Matriz de confusión:**

***Figura 58***

*Matriz de confusión modelo Random forest df_submuestreo*




El modelo si tuvo un mejor rendimiento y balance entre las clases, tuvo un buen rendimiento en la clase POS donde tuvo un resultado en recall de 94% y en F1-score de 91%. El índice Gini también tuvo un buen resultado (0.7772) lo que indico una buena capacitad de pureza.

***Implementación de mejora al modelo df_submuestreo***

Para mejorar mucho más el modelo con el entrenamiento de df_submuestreo, se mejoró y se hizo algunos cambios (Ver Anexo – Modelado Random Forest df_submuestreo mejorado.):

- **min_df=3 y max_df=0.95** Se controló la frecuencia mínima y máxima.

- **ngram_range= (1, 2)** Unigramas y bigramas.
- **sublinear_tf=True** Se ajustó logarítmicamente la frecuencia.
- **max_features=10000** Se limitó el tamaño de características de las 10.000 características más relevantes.
- **n_estimators:** Número de árboles (200, 300, 500).
- **max_depth:** Profundidad máxima (30, 50 o sin límite).
- **min_samples_split:** Número mínimo de muestras para dividir un nodo (2 o 5).
- **min_samples_leaf:** Número mínimo de muestras por hoja (1, 3 o 5).
- **max_features:** Selección de características (sqrt o log2).
- **class_weight:** 'balanced' se compenso el desbalance residual entre clases.

**Evaluación del modelo Random Forest:**

***Tabla 20***

*Evaluación modelo Random Forest df_submuestreo mejorado*

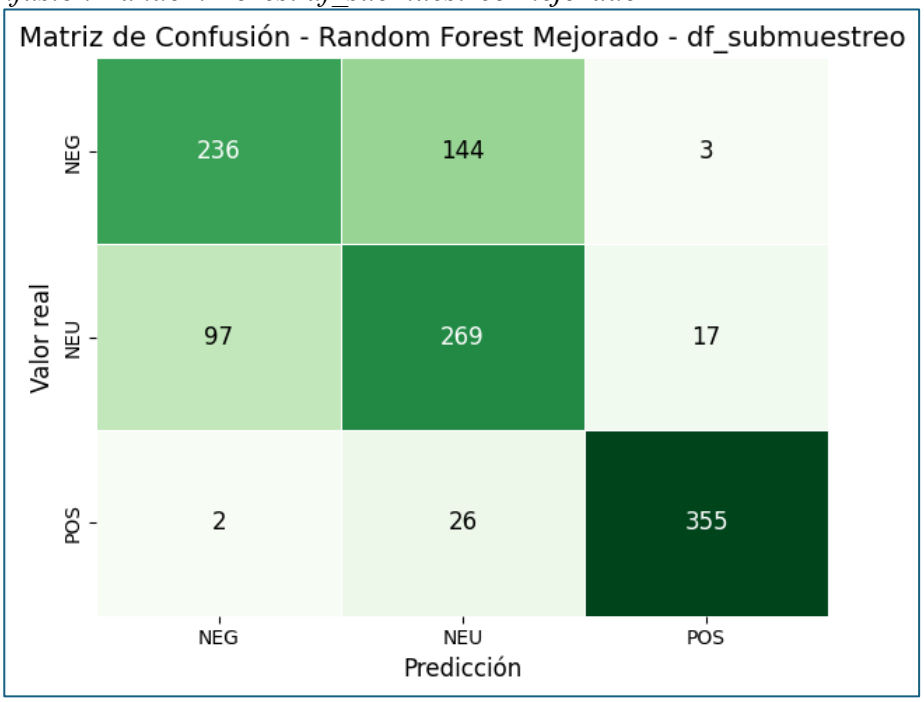
| | *Precision* | *Recall* | *F1-score* | *Support* |
|---|---|---|---|---|
| ***NEG*** | 0.70 | 0.62 | 0.66 | 383 |
| ***NEU*** | 0.61 | 0.70 | 0.65 | 383 |
| ***POS*** | 0.95 | 0.93 | 0.94 | 383 |
| ***Accuracy*** | | | 0.75 | 1149 |

| | | | | |
|---|---|---|---|---|
| ***Macro avg*** | 0.75 | 0.75 | 0.75 | 1149 |
| ***Weighted avg*** | 0.75 | 0.75 | 0.75 | 1149 |
| ***Gini Index*** | | | | 0.7886 |

**Matriz de Confusión:**

***Figura 59***

*Matriz de confusión Random Forest df_submuestreo mejorado*



Se obtuvo buenos resultados entre las clases. En la clase POS se obtuvo buena precisión con 95% y una sensibilidad de 93%. Hubo un balance entre clases NEG y NEU. El índice Gini también fue superior ya que subió a 0,78 su pureza.

***Resumen de evaluación algoritmo Random Forest.***

- **1 modelo: Random Forest - df_final (sin mejoras)**
    - Tuvo un gran sesgo con las 3 clases NEG, NEU y POS.
- **2 modelo: Random Forest - df_final (mejorado)**
    - Mejoró el recall para las clases NEU y POS,
    - Se mantuvo un gran sesgo hacia la clase NEG.
- **3 modelo: Random Forest - df_submuestreo (sin mejora)**
    - Fue más equilibrado
    - Clasifica bien la clase positiva **(POS) y (NEU)**.
- **4 modelo: Random Forest - df_submuestreo (mejorado)**
    - Mejoró el modelo en general.
    - La precisión y recall mejoró en las 3 clases.
    - Clasifica mejor la clase POS.

***Tabla de comparación modelos Random Forest:***

***Tabla 21***

*Comparación de modelos Random Forest*

| ***Modelo*** | ***Accuracy*** | ***F1 Macro*** | ***Gini*** | ***Bueno*** |
|---|---|---|---|---|
| ***RF - df_final*** | 0.66 | 0.51 | 0.5213 | NEG (únicamente) |
| ***RF Mejorado - df_final*** | 0.67 | 0.57 | 0.5484 | NEG, mejoro NEU/POS |
| ***RF - df_submuestreo*** | 0.74 | 0.74 | 0.7772 | POS, balanceado en general |
| ***RF - df_submuestreo mejorado*** | 0.75 | 0.75 | 0.7886 | Mejor resultado global |

**Conclusión:**

El mejor modelado para Random Forest fue df_submuestreo mejorado debido a que reconoció las 3 clases adecuadamente, tienen un mayor índice de Gini, la clase POS se detectó mejor y con buena precisión con las métricas precisión y recall.

3. **Algoritmo: SVM**

- ***Entrenamiento con DataSet df_final***

Ahora se entrenó el df_final con SVM (Maquinas de soporte vectorial) se usó LinearSV, este fue eficiente en tareas de clasificación de texto y manejo bien el texto con alta dimensión y tuvo buena eficiencia computacional (Ver Anexo – Modelado SVM df_final.).

Este entrenamiento, se utilizó algunas configuraciones, que se muestran a continuación:

- **min_df=5 y max_df=0.95** Eliminar términos poco raros o palabras muy frecuentes.
- **ngram_range=(1, 2)** Unigramas y bigramas.
- **sublinear_tf=True** Se aplico la escala logarítmica a frecuencias

**Evaluación de modelo SVM:**

***Tabla 22***
*Evaluación de modelo SVM df_final*

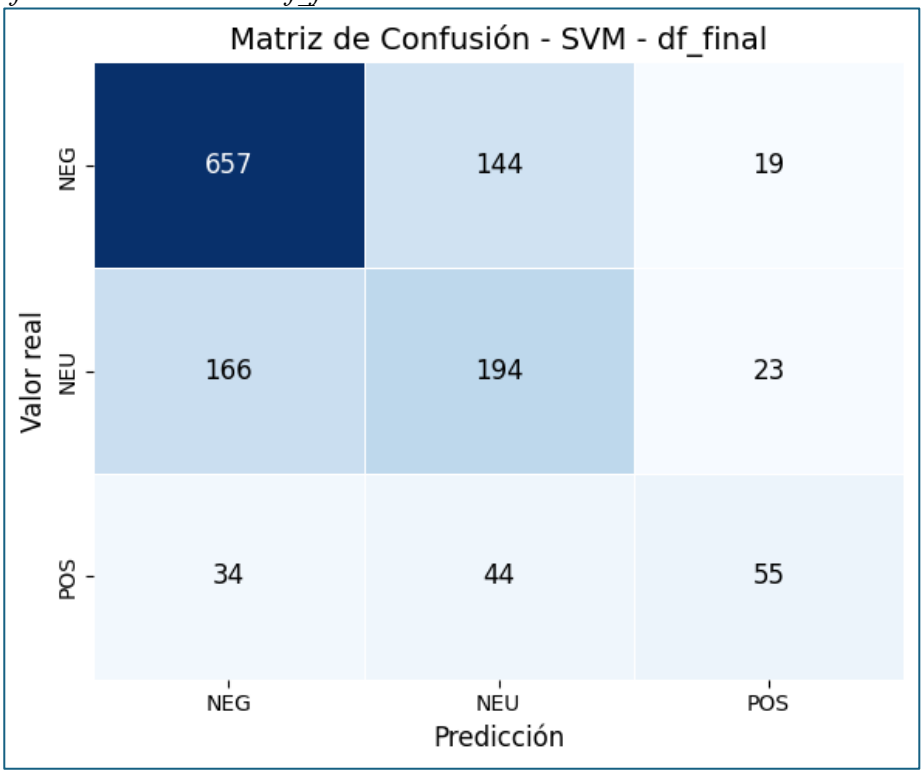
| | *Precision* | *Recall* | *F1-score* | *Support* |
|---|---|---|---|---|
| ***NEG*** | 0.77 | 0.80 | 0.78 | 820 |
| ***NEU*** | 0.51 | 0.51 | 0.51 | 383 |
| ***POS*** | 0.57 | 0.41 | 0.48 | 133 |
| ***Accuracy*** | | | 0.68 | 1336 |

| | | | | |
|---|---|---|---|---|
| ***Macro avg*** | 0.61 | 0.57 | 0.59 | 1336 |
| ***Weighted avg*** | 0.67 | 0.68 | 0.67 | 1336 |
| ***distance to hyperplane*** | *Mín: -3.1745* | | Máx: 3.5484 | |

**Matriz de confusión:**

***Figura 60***

*Matriz de confusión modelo SVM df_final*




Después de haber realizado el entrenamiento, se observó que hubo un buen rendimiento en la clase NEG, alcanzando un puntaje en el F1-score de 78%. Con respecto al desempeño en

la clase NEU y POS, no fue bueno, pero fue aceptable, esto se dio debido al desbalance de los datos y también a la ambigüedad entre las clases.

### *Implementación de mejora al modelo df_final*

Para mejorar el rendimiento de SVM con el DataSet df_final se realizó un proceso de ajuste de hiperparámetros con validación cruzada. Se uso el algoritmo de SVC de Sklearn el cual ayudó (Ver Anexo – Modelado SVM df_final mejorado.).

Las técnicas y configuraciones que se aplicaron fueron las siguientes:

- **min_df=5 y max_df=0.95:** Filtrar términos poco frecuentes o de alta frecuencia.
- **ngram_range=(1, 2):** Unigramas y bigramas.
- **sublinear_tf=True:** Se aplico el escalado logarítmico a las frecuencias.
- Regularización de hiperparámetros C (0.1, 1, 10).
- Se uso dos tipos de kernel: 'linear' y 'rbf'.
- Se balanceo la clase con **class_weight='balanced'.**
- Se ajustó el parámetro gamma ('scale' y 'auto') en el kernel 'rbf',

**Evaluación del modelo SVM:**

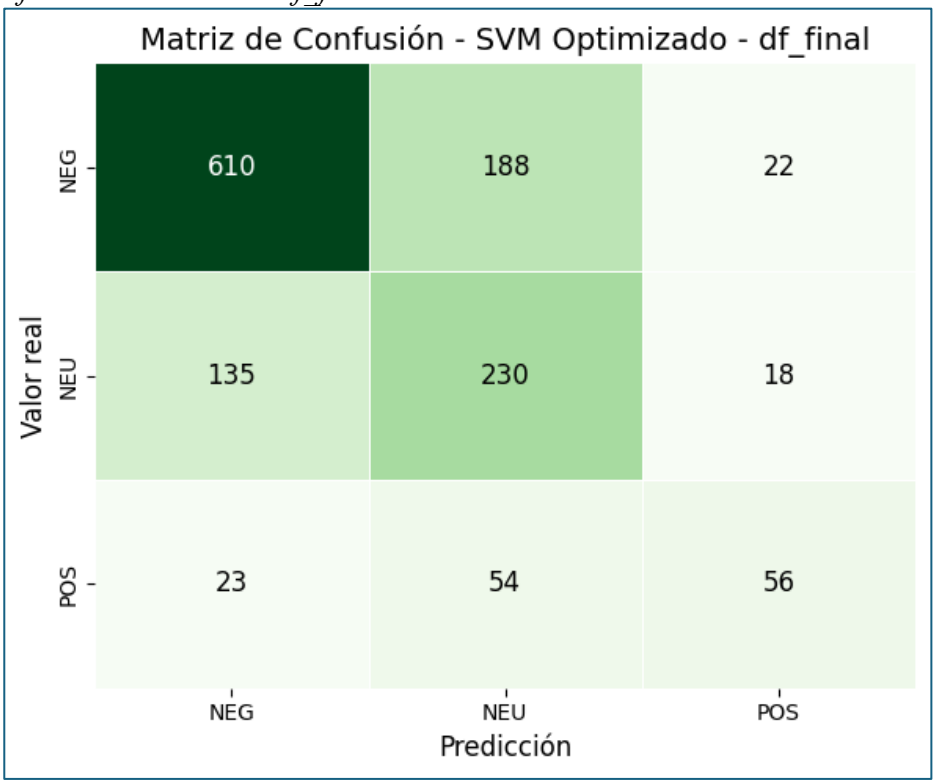
***Tabla 23***

*Evaluación modelo SVM df_final mejorado.*

| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.79 | 0.74 | 0.77 | 820 |
| ***NEU*** | 0.49 | 0.60 | 0.54 | 383 |
| ***POS*** | 0.58 | 0.42 | 0.49 | 133 |
| ***Accuracy*** | | | 0.67 | 1336 |
| ***Macro avg*** | 0.62 | 0.59 | 0.60 | 1336 |
| ***Weighted avg*** | 0.69 | 0.67 | 0.67 | 1336 |
| ***distance to hyperplane*** | *Mín: -0.2562* | | Máx: 2.2705 | |

***Figura 61***

*Matriz de confusión modelo SVM df_final actualizado*




Este entrenamiento si mejoro a diferencia del anterior, se observó alguna mejoría en las clases NEU y POS. El uso del Kernel no lineal si ayudo a captar las relaciones más complicadas, lo cual trajo consigo una mejor distribución en el rendimiento del modelo. Este buen balance sirvió para mitigar el sesgo que había hacia la clase NEG la cual es la mayoritaria.

- ***Entrenamiento con DataSet df_submuestreo***

Se trabajó con el entrenamiento de SVM con la data de df_submuestreo. Esta data ayudó a mitigar el sesgo que pueda aparecer dentro del entrenamiento hacia la clase NEG (Ver Anexo – Modelado SVM df_submuestreo.). La configuración que se usó para este caso fueron las siguientes:

- **min_df=5 y max_df=0.95:** filtrar términos poco representativos.

- **ngram_range=(1, 2):** para incluir unigramas y bigramas.
- **sublinear_tf=True:** Se aplicó escalado logarítmico a las frecuencias.

**Evaluación del modelo SVM:**

***Tabla 24***

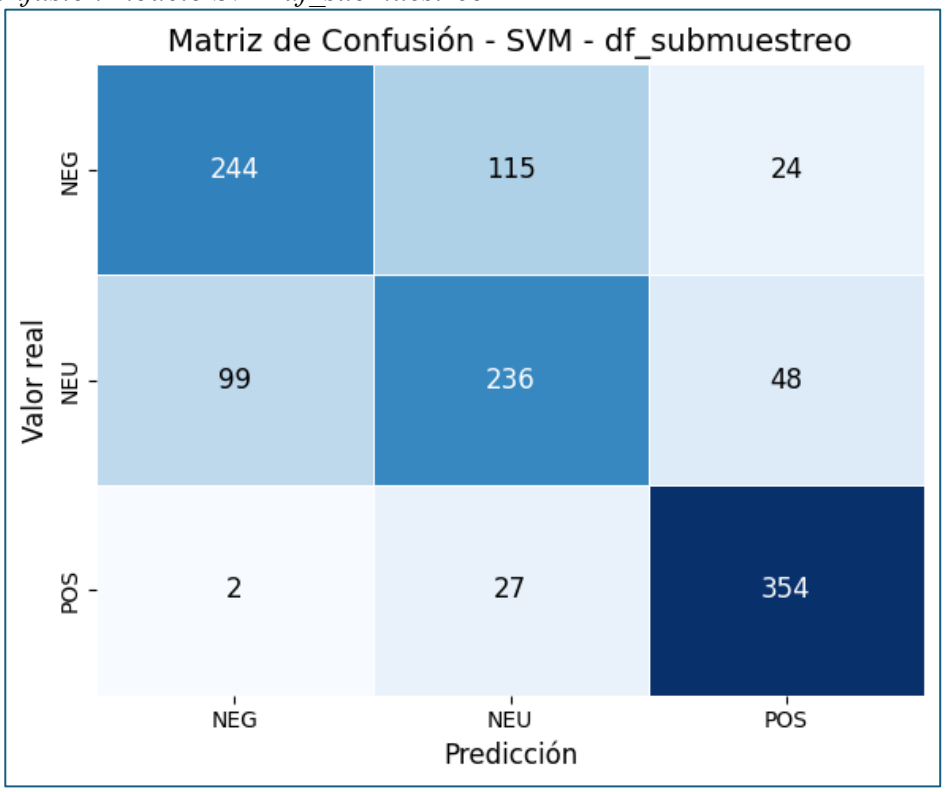
*Evaluación modelo SVM df_submuestreo*

| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.71 | 0.64 | 0.67 | 383 |
| ***NEU*** | 0.62 | 0.62 | 0.62 | 383 |
| ***POS*** | 0.83 | 0.92 | 0.88 | 383 |
| ***Accuracy*** | | | 0.73 | 1149 |
| ***Macro avg*** | 0.72 | 0.73 | 0.72 | 1149 |
| ***Weighted avg*** | 0.72 | 0.73 | 0.72 | 1149 |
| ***distance to hyperplane*** | *Mín: -3.1597* | | Máx: 2.5614 | |

**Matriz de confusión:**

***Figura62***

*Matriz de confusión modelo SVM df_submuestreo*




Con este entrenamiento se obtuvo un mejor rendimiento con la clase POS, con un puntaje de F1-Score de 88%. Hubo un buen equilibrio de clases y esto ayudó a que haya un buen desempeño como el que se tuvo en la clase NEU que ahora subió su calificación. La clase NEG disminuyo, pero al final el resultado fue más equilibrado entre las 3 clases.

***Implementación de mejora al modelo df_submuestreo***

En base a los resultados del modelado anterior, se hizo un ajuste para mejorar el modelado, se usó GridSearch para una combinación amplia (Ver Anexo – Modelado SVM

df_submuestreo mejorado.). donde se incorporó una optimización en los hiperparámetros siguientes:

- **min_df=3:** Alta sensibilidad en términos informativos con una frecuencia baja.
- **max_df=0.95:** filtrado de términos demasiado comunes.
- **ngram_range= (1, 2):** Unigramas y bigramas.
- **sublinear_tf=True:** Se escaló logarítmicamente la frecuencia.
- **max_features=15000:** ampliación de la cantidad de características permitidas.

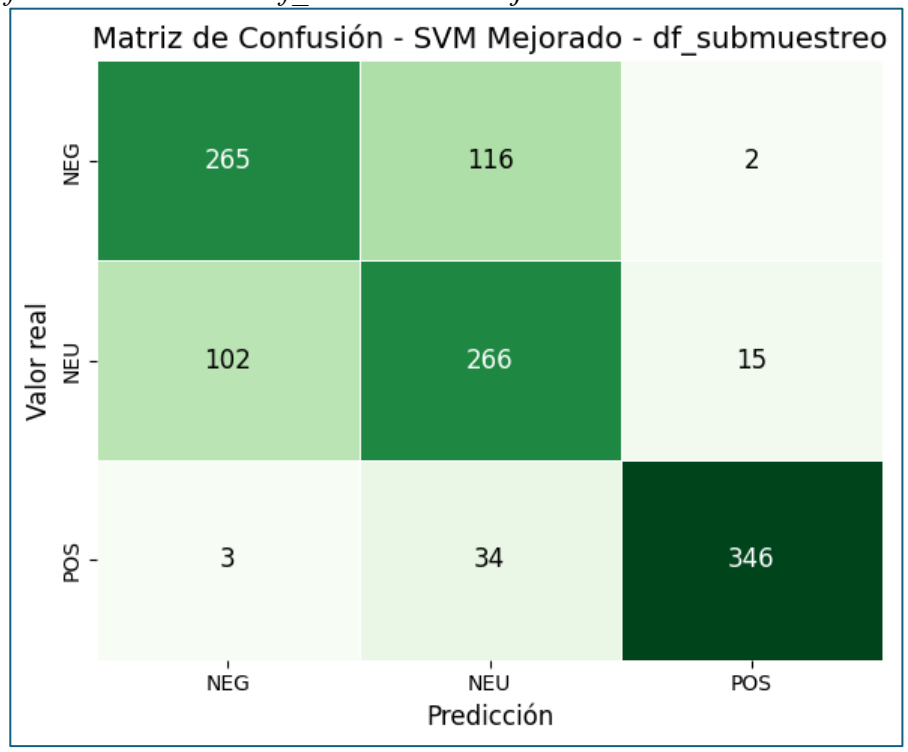
***Tabla 25***

*Evaluación modelo SVM df_submuestreo mejorado*

| | *Precision* | *Recall* | *F1-score* | *Support* |
|---|---|---|---|---|
| ***NEG*** | 0.72 | 0.69 | 0.70 | 383 |
| ***NEU*** | 0.64 | 0.69 | 0.67 | 383 |
| ***POS*** | 0.95 | 0.90 | 0.93 | 383 |
| ***Accuracy*** | | | 0.76 | 1149 |
| ***Macro avg*** | 0.77 | 0.76 | 0.77 | 1149 |
| ***Weighted avg*** | 0.77 | 0.76 | 0.77 | 1149 |
| ***distance to hyperplane*** | *Mín: -0.2625* | | Máx: 2.2642 | |

***Figura 63***

*Matriz de confusión modelo SVM df_submuestreo mejorado*




Este último modelado tuvo un mejor equilibrio en general en las 3 clases. Este, destacó en la clase POS, sin dejar tan sesgado a esta clase ya que los resultados y precisión de las clases NEG y NEU son buenas también.

***Resumen de evaluación algoritmo SVM.***

- **1 modelo: SVM - df_final (sin mejoras)**
    - Tuvo un mejor rendimiento solo con la clase NEG.
    - Las clases NEU y POS están desbalanceadas y tienen unos puntajes bajo sen recall y precisión.
    - Si bien hubo un margen largo de hiperplano, pero fue porque el modelo clasifica a la segura la clase NEG.

- **2 modelo: SVM - df_final (mejorado)**
    - Hubo una pequeña mejora para reconocer la clase NEU.
    - Hubo un margen corto de distancia de hiperplano por lo que pudo generar errores en la clasificación
    - Mejoró un poco la clase POS, pero aún hay debilidades para clasificar esta clase.
- **3 modelo: SVM - df_submuestreo (sin mejora)**
    - Mejoró considerablemente la clasificación de la clase POS.
    - Las 3 clases se equilibraron mucho mejor en el recall y precisión.
    - Obtuvo un buen rendimiento, pero se mostró mejor hacia la clase POS.
    - Fue más preciso en general pero todavía hay comentarios mal clasificados.
- **4 modelo: SVM - df_submuestreo (mejorado)**
    - Mejoró en precisión, recall y F1-Score.
    - Tuvo una excelente precisión en la clase POS.
    - Mejoró el recall en la clase NEU.

***Tabla de comparación modelos SVM:***

***Tabla 26***

*Comparación de modelos SVM*

| *Modelo* | *Accuracy* | *F1 Macro* | *Bueno* |
|---|---|---|---|
| ***SVM - df_final*** | 0.68 | 0.59 | NEG |
| ***SVM Mejorado - df_final*** | 0.67 | 0.60 | NEG, ligeramente NEU |
| ***SVM - df_submuestreo*** | 0.73 | 0.72 | POS y NEU |
| ***SVM - df_submuestreo mejorado*** | 0.76 | 0.77 | Mayor balance (POS, NEG, NEU) |

**Conclusión:**

El mejor modelado para SVM fue el mejorado con df_submuestreo, debido a que obtuvo un buen F1-score para todas las clases, también mejoro la precisión, recall y hay un mejor balanceo de clases.

4. **Algoritmo: XGBoost**

- ***Entrenamiento con DataSet df_final***

Para finalizar con los algoritmos tradicionales se entrenará y se observara el rendimiento del modelado con XGBoost, para entrenar este modelo, se empezó con la configuración siguiente (Ver Anexo – Modelado XGBoost df_final.):

- Vectorización con TF-IDF
- **objective='multi:softprob'**, regresa las probabilidades a cada clase.
- **eval_metric='mlogloss'**: ayuda a mitigar o eliminar la entropía cruzada (log loss).
- **use_label_encoder=False y random_state=42** reproducibilidad y evitar advertencias.

**Evaluación del modelo XGBoost:**

***Tabla 27***
*Evaluación del modelo XGBoost df_final.*

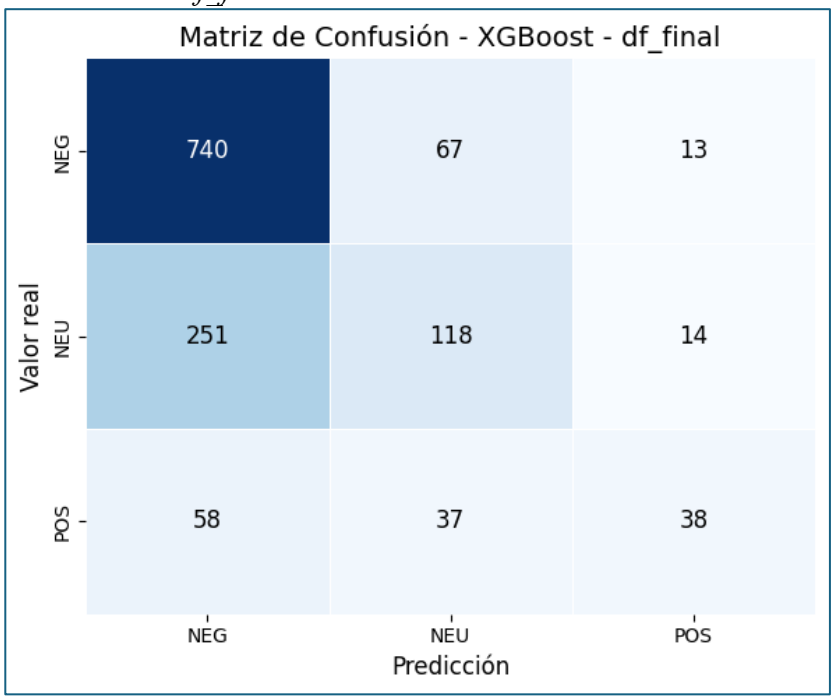
| | *Precision* | *Recall* | *F1-score* | *Support* |
|---|---|---|---|---|
| ***NEG*** | 0.71 | 0.90 | 0.79 | 820 |
| ***NEU*** | 0.53 | 0.31 | 0.39 | 383 |
| ***POS*** | 0.58 | 0.29 | 0.38 | 133 |
| ***Accuracy*** | | | 0.67 | 1336 |
| ***Macro avg*** | 0.61 | 0.50 | 0.52 | 1336 |

| | | | | |
|---|---|---|---|---|
| ***Weighted avg*** | 0.64 | 0.67 | 0.64 | 1336 |
| ***Log loss*** | | | | 0.7092 |

**Matriz de confusión:**

***Figura 64***

*Matriz de confusión XGBoost df_final*




El modelo XGBoost tuvo un gran rendimiento con las clases NEG, pero un rendimiento bajo en las clases NEU y POS, la métrica usada, log loss (0.7092), ayuda a ver que el modelo tiene incertidumbre al momento de clasificar entre las clases.

***Implementación de mejora al modelo df_final.***

Para mejorar el modelo anterior se implementaron algunos ajustes en los hiperparámetros (Ver Anexo – Modelado XGBoost df_final mejorado), a continuación, se presenta los ajustes realizados:

- Vectorización con (TF-IDF)
- **max_depth:** profundidad (entre 3 y 7),
- **learning_rate:** tasa de aprendizaje (entre 0.01 y 0.3),
- **n_estimators:** número de árboles (entre 100 y 300),
- **subsample:** datos utilizados por cada árbol (0.8 a 1.0),
- **colsample_bytree:** características utilizadas por árbol (0.8 a 1.0).

**Evaluación del modelo XGBoost:**

***Tabla 28***
*Evaluación modelo XGBoost df_final mejorado*

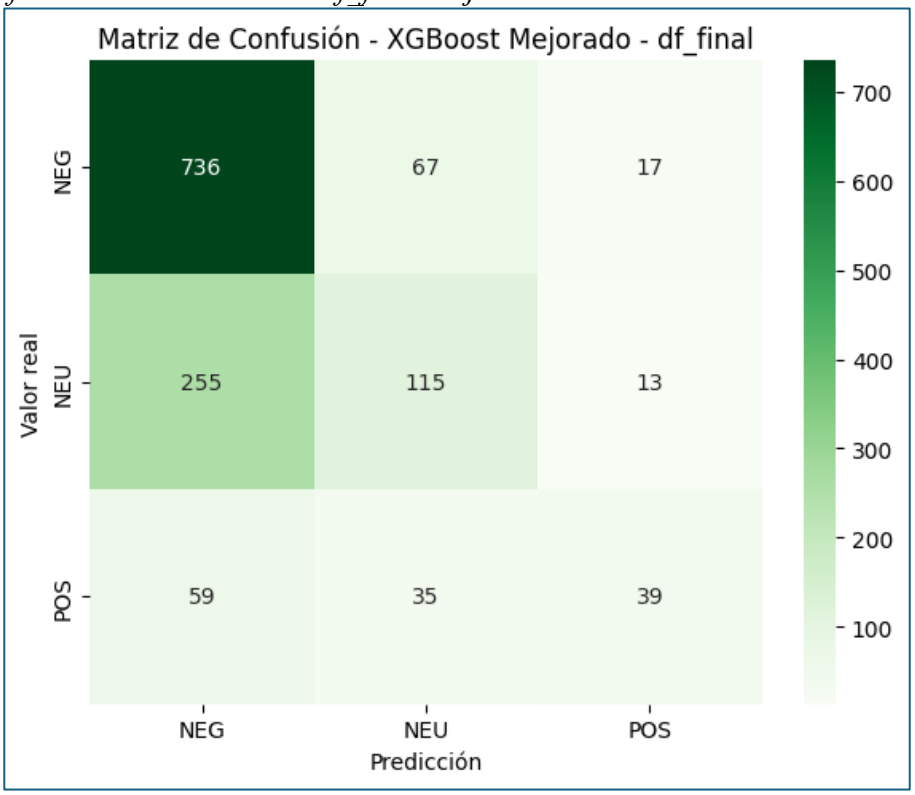
| | *Precision* | *Recall* | *F1-score* | *Support* |
|---|---|---|---|---|
| ***NEG*** | 0.70 | 0.90 | 0.79 | 820 |
| ***NEU*** | 0.53 | 0.30 | 0.38 | 383 |
| ***POS*** | 0.57 | 0.29 | 0.39 | 133 |
| ***Accuracy*** | | | 0.67 | 1336 |

| | | | | |
|---|---|---|---|---|
| ***Macro avg*** | 0.60 | 0.50 | 0.52 | 1336 |
| ***Weighted avg*** | 0.64 | 0.67 | 0.63 | 1336 |
| ***Log loss*** | | | | 0.7093 |

**Matriz de confusión:**

***Figura 65***

*Matriz de confusión modelo XGBoost df_final mejorado*



Se obtuvo una ligera mejora a comparación del modelo anterior, ayudo a mejorar la clase NEG, pero las clases NEU y POS siguieron siendo muy bajas a diferencia de NEG. La precisión general fue 67%.

- ***Entrenamiento con DataSet df_submuestro***

Para mejorar el modelado, se entrenó con el DataSet balanceado ya que aquí hay equilibrio en las 3 clases y se pudo evitar el sesgo en la clase NEG (Ver Anexo – Modelado XGBoost df_submuestreo.). Las configuraciones ayudaron a tener un mejor desempeño en general.

**Evaluación del modelo XGBoost:**

***Tabla 29***

*Evaluación de modelo XGBoost df_submuestreo*

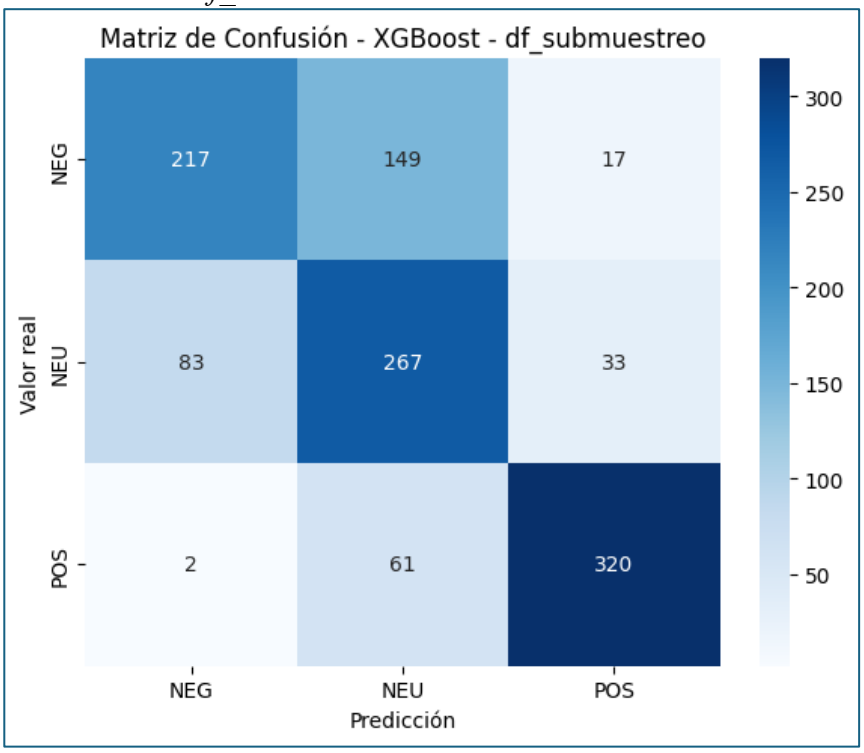
| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.72 | 0.57 | 0.63 | 383 |
| ***NEU*** | 0.56 | 0.70 | 0.62 | 383 |
| ***POS*** | 0.86 | 0.84 | 0.85 | 383 |
| ***Accuracy*** | | | 0.70 | 1149 |
| ***Macro avg*** | 0.71 | 0.70 | 0.70 | 1149 |

| | | | | |
|---|---|---|---|---|
| ***Weighted avg*** | 0.71 | 0.70 | 0.70 | 1149 |
| ***Log loss*** | | | | 0.7020 |

**Matriz de confusión:**

***Figura 66***

*Matriz de confusión XGBoost df_submuestreo*




El rendimiento con df_submuestreo fue mucho mejor ya que fue más equilibrado a diferencia con los datos reales. Detecto muy bien la clase POS con un F1-score 85% y tuvo una mejora en la clase NEU. El uso de los datos balanceados si influencio en la mejora del modelo.

***Implementación de mejora al modelo df_submuestreo.***

Para mejorar el modelo anterior que se entrenó con df_submuestreo, se usó el vectorizador para transformar los comentarios, se aplicó una búsqueda con randomized search donde se exploró 30 combinaciones por medio de validación cruzada 3 pliegues y se realizó las siguientes configuraciones (Ver Anexo – Modelado XGBoost df_submuestreo mejorado):

- **max_depth:** profundidad máxima de los árboles.
- **learning_rate:** tasa de aprendizaje.
- **n_estimators:** número de árboles.
- **subsample:** muestras utilizadas por cada árbol.
- **colsample_bytree:** columnas usadas por árbol.

**Evaluación del modelo XGBoost:**

***Tabla 30***

*Evaluación Matriz modelo XGBoost df_submuestreo mejorado.*

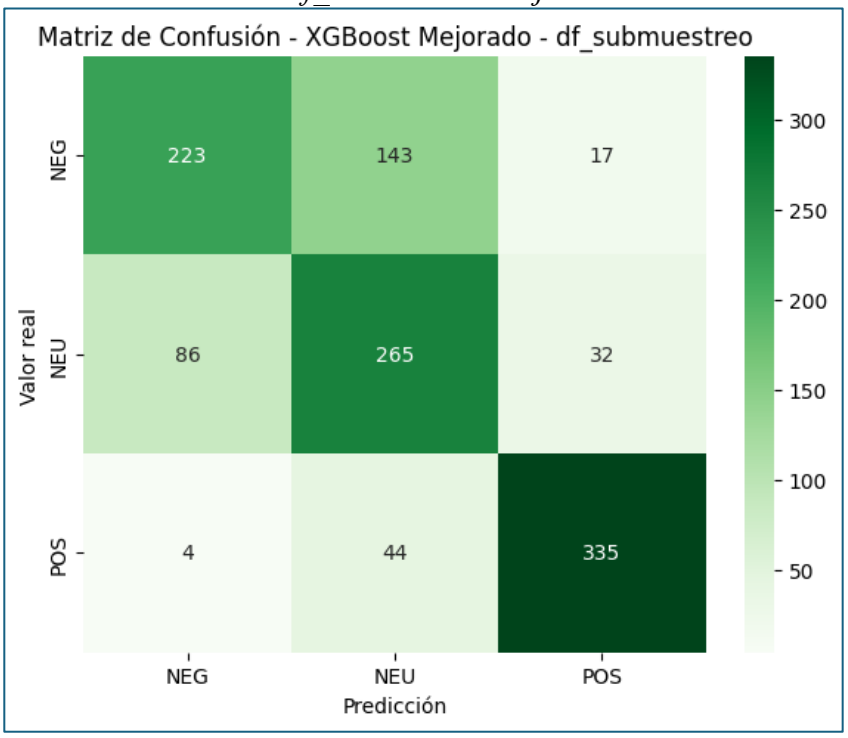
| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.71 | 0.58 | 0.64 | 383 |
| ***NEU*** | 0.59 | 0.69 | 0.63 | 383 |
| ***POS*** | 0.87 | 0.87 | 0.87 | 383 |
| ***Accuracy*** | | | 0.72 | 1149 |

| | | | | |
|---|---|---|---|---|
| ***Macro avg*** | 0.72 | 0.72 | 0.72 | 1149 |
| ***Weighted avg*** | 0.72 | 0.72 | 0.72 | 1149 |
| ***Log loss*** | | | | 0.6732 |

**Matriz de confusión:**

***Figura 67***

*Matriz de confusión modelo XGBoost df_submuestreo mejorado*




Esta mejora ayudo a mejorar ligeramente el modelo, se observó que hubo un mejor balance entre clases, donde la clase NEU tuvo un mejor rendimiento y la clase POS también tuvo un desempeño bueno.

***Resumen de evaluación algoritmo XGBoost.***

- **1 modelo: XGBoost - df_final (sin mejoras)**
    - Tuvo un mejor rendimiento en la clase NEG.
    - Las clases NEU y POS están desbalanceadas y tienen puntajes bajos.
    - Log Loss esta elevado (0.7092), lo que indicó una incertidumbre en las predicciones de clases.
    - Hubo un sesgo en la clase NEG que era la clase mayoritaria en el DataSet desbalanceado.
- **2 modelo: XGBoost - df_final (mejorado)**
    - No hubo una mejora como tal debido a que el F1 macro y el log loss son casi idénticos al anterior modelo.
    - La mejora no ayudo mucho debido al gran desbalance entre clases.
    - No fue recomendable seguir mejorando este modelo por el desbalanceo.
- **3 modelo: XGBoost - df_submuestreo (sin mejora)**
    - Mejoró el rendimiento de la clase NEU y POS en un valor considerable.
    - El modelo se equilibró mejor en las 3 clases, con un resultado de (F1=0.85) para la clase POS.
    - Log Loss también bajo a 0.7020 lo que se puede decir que hay predicciones más seguras.
- **4 modelo: XGBoost - df_submuestreo (mejorado)**
    - Mejoró en las métricas:
        - F1 macro = 0.72
        - F1 POS = 0.87, F1 NEU = 0.63, F1 NEG = 0.64

- Log Loss = (0.6732)

- Hay buen balanceo entre clases y estabilidad.

***Tabla de comparación modelos XGBoost:***

***Tabla 31***

*Comparación de modelos XGBoost*

| *Modelo* | *Accuracy* | *F1 Macro* | *Log Loss* | *Bueno* |
|---|---|---|---|---|
| **XGBoost - *df_final*** | 0.67 | 0.52 | 0.7092 | NEG |
| **XGBoost *Mejorado - df_final*** | 0.67 | 0.52 | 0.7093 | NEG |
| **XGBoost - *df_submuestreo*** | 0.70 | 0.70 | 0.7020 | POS y NEU |
| **XGBoost - *df_submuestreo mejorado*** | 0.72 | 0.72 | 0.6732 | Mayor balance (POS, NEG, NEU) |

**Conclusión:**

El mejor modelado para XGBoost fue el mejorado con df_submuestreo, debido a que obtuvo un buen resultado para todas las clases a diferencia de los demás, también mejoro el puntaje de log loss en cual bajo a 0.6732.

## 5. Modelo LLM ChatGpt

Se implemento el uso de modelos LLMs para ver su rendimiento en la clasificación de los comentarios, en este caso, se usó ChatGpt desarrollado por OpenAI.

A diferencia de los demás entrenamientos que requieren un entrenamiento previo, este fue evaluado directamente gracias a su gran capacidad semántica, se usó el modelo GPT-3.5-turbo) para realizar esta clasificación. Se uso la técnica zero-shot learning, donde la respuesta es en base a una instrucción dada.

Cada comentario fue enviado con la siguiente instrucción:

- "Clasifica el sentimiento del siguiente comentario como POS (positivo), NEG (negativo) o NEU (neutral). Solo devuelve POS, NEG o NEU: '[comentario]"

Se uso la librería tqdm para ver el avance y tiempo que toma él envió de cada petición por comentario (Ver Anexo – Modelado ChatGpt df_final.).

- ***Entrenamiento con DataSet df_final***

**Tiempo de ejecución:** 74 minutos, 26.2 segundos.

**Evaluación del modelo ChatGpt:**

***Tabla 32***

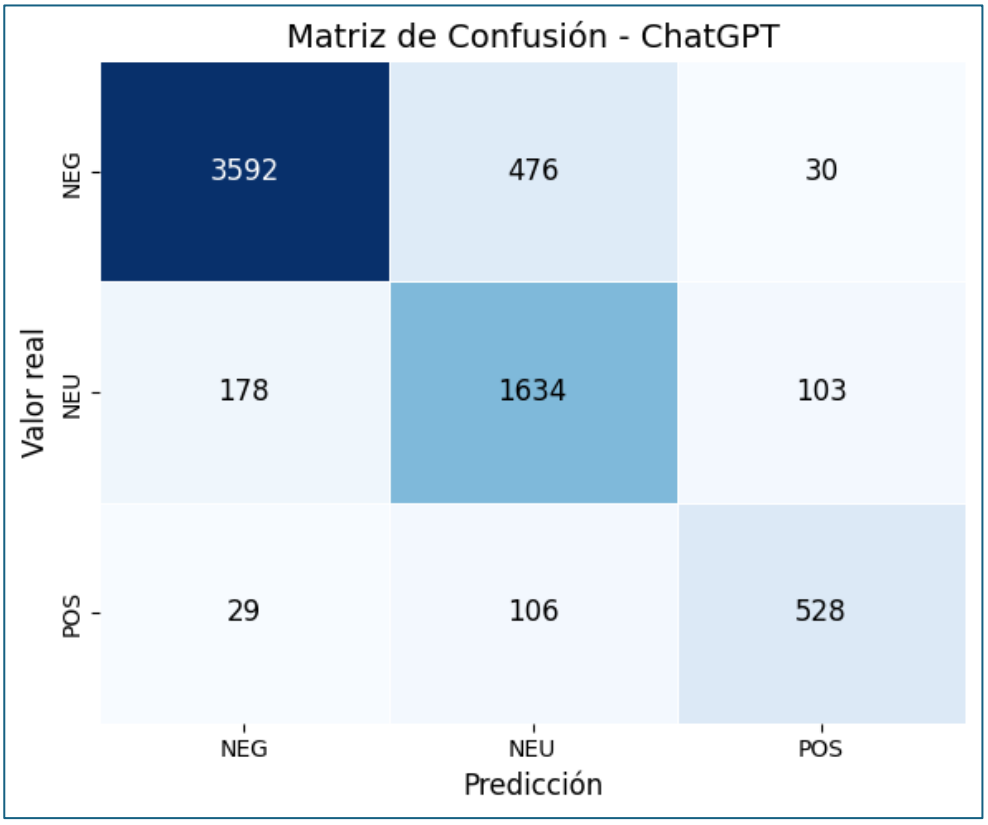
*Evaluación Modelo ChatGpt df_final*

| | *Precision* | *Recall* | *F1-score* | *Support* |
|---|---|---|---|---|
| ***NEG*** | 0.95 | 0.88 | 0.91 | 4098 |
| ***NEU*** | 0.74 | 0.85 | 0.79 | 1915 |
| ***POS*** | 0.80 | 0.80 | 0.80 | 663 |
| ***Accuracy*** | | | 0.86 | 6676 |
| ***Macro avg*** | 0.83 | 0.84 | 0.83 | 6676 |
| ***Weighted avg*** | 0.87 | 0.86 | 0.86 | 6676 |
| ***Inference Time*** | | | | 4466.00s |
| ***Avg time per comment*** | | | | 0.67s |

**Matriz de confusión:**

***Figura 68***

*Matriz de confusión modelo ChatGpt df_final*



**Conclusión:**

Demostró un gran desempeño en la clasificación de las 3 clases, su clase más fuerte fue la clase NEU, donde la mayoría de los otros modelos fallaron. El recall y precisión para la clase POS, fue alto. La velocidad por comentario fue buena (0.65 por comentario). Algo a destacar también fue el alto accuracy y macro promedio general, fue el mejor de todos los modelos.

## 6. Modelo LLM DeepSeek

También se implementó el modelo de DeepSeek, similar a ChatGpt, el cual usa tareas de comprensión o clasificación de texto. Su integración ayudo para revisar el rendimiento entre estas dos inteligencias y los algoritmos tradicionales.

Al igual que ChatGpt se usó la técnica de zero-shot mediante la misma instrucción o prompt, donde no fue necesario entrenar al modelo previamente, donde solo se realizó la consulta a la API de DeepSeek, para clasificar los comentarios en nuestras clases (POS, NEG, NEU). Para este caso se implementó paralelismo con ThreadPoolExecutor que ayudo al procesamiento donde se usó hasta 5 threads a la vez. Como se manejó procesamiento en paralelo, se implementó tolerancia a fallos donde se podía reintentar él envió de la petición hasta 3 intentos por comentario (Ver Anexo – Modelado DeepSeek df_final.).

- ***Entrenamiento con DataSet df_final***

**Tiempo de ejecución:** 225 minutos, 13.1 segundos.

**Evaluación del modelo DeepSeek:**

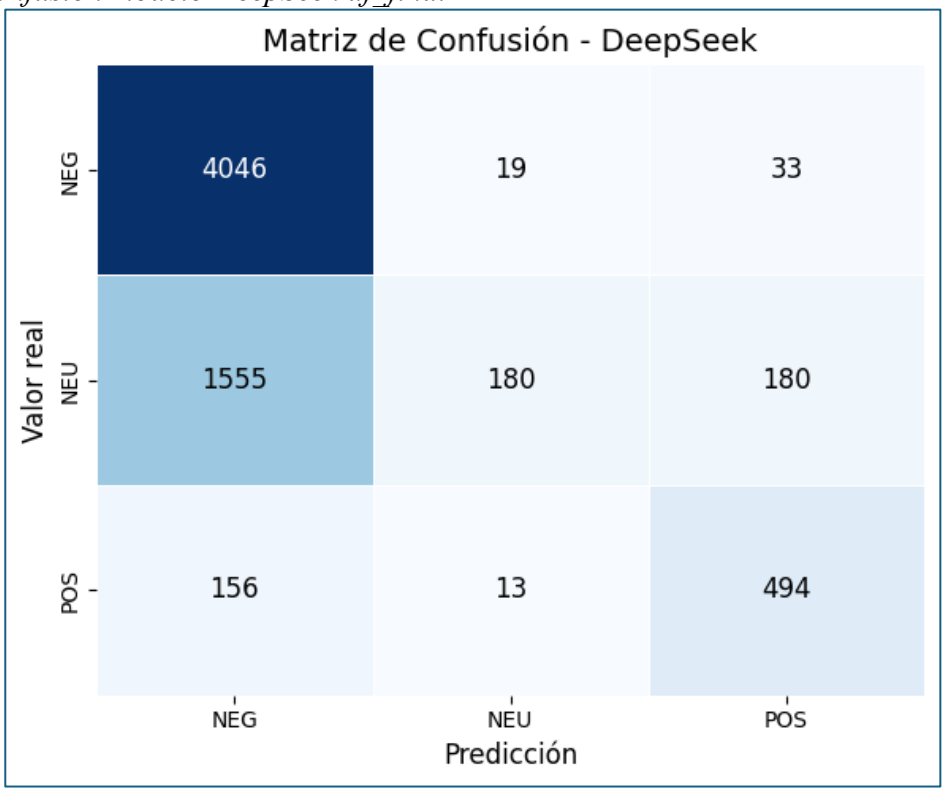
***Tabla 33***
*Evaluación modelo DeepSeek df_final*

| | ***Precision*** | ***Recall*** | ***F1-score*** | ***Support*** |
|---|---|---|---|---|
| ***NEG*** | 0.70 | 0.99 | 0.81 | 4098 |
| ***NEU*** | 0.85 | 0.09 | 0.17 | 1915 |
| ***POS*** | 0.70 | 0.75 | 0.72 | 663 |
| ***Accuracy*** | | | 0.71 | 6676 |
| ***Macro avg*** | 0.75 | 0.61 | 0.57 | 6676 |
| ***Weighted avg*** | 0.74 | 0.71 | 0.62 | 6676 |
| ***Inference Time*** | | | | 13514.00s |
| ***Avg time per comment*** | | | | 2.02s |

**Matriz de confusión:**

***Figura 69***

*Matriz de confusión modelo DeepSeek df_final*




**Conclusión:**

Sesgado a la clase NEG y la mayoría de los comentarios NEU fueron mal clasificados con clase NEG. Si tuvo un buen desempeño en la clase POS, pero no supero a ChatGpt. El recall en la clase NEU fue super malo ya que solo obtuvo el 0.09. El tiempo de respuesta también es super alto, 3 veces más lento que ChatGpt.

**Comparación de LLMs**

***Tabla 34***

*Tabla comparación de modelos LLMs*

| *Modelo* | *Accuracy* | *F1-Macro* | *F1-NEU* | *F1-POS* | *Tiempo por comentario* | *Mejor* |
|---|---|---|---|---|---|---|
| ***ChatGPT*** | 0.86 | 0.83 | 0.79 | 0.80 | 0.67s | En todas las clases |
| ***DeepSeek*** | 0.71 | 0.57 | 0.17 | 0.72 | 2.02s | NEG (pero mal NEU/POS) |

## 4.3 Evaluación de modelos de análisis de sentimientos, incluidos modelos (LLMs)

**FASE 5: Evaluación del modelo**

***Evaluar resultados***

**Comparación final de los 6 modelos:**

***Tabla 35***

*Comparación final de modelos*

| *Modelo* | *Accuracy* | *F1-Macro* | *F1-POS* | *F1-NEU* | *F1-NEG* | *Tiempo por comentario* | *Comentarios Clave* |
|---|---|---|---|---|---|---|---|
| ***ChatGPT*** | 0.86 | 0.83 | 0.80 | 0.79 | 0.91 | 0.67s | Balance y precisión excelentes en las 3 clases. |
| ***SVM Mejorado - df_submuestreo*** | 0.76 | 0.77 | 0.93 | 0.67 | 0.70 | | Mejor modelo tradicional, clasificó bien la clase POS |
| ***XGBoost*** | 0.72 | 0.72 | 0.82 | 0.63 | 0.64 | | Ideal solo para tareas para |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | comentarios NEU y POS. |
| ***Random Forest Mejorado - df_sub*** | 0.75 | 0.75 | 0.94 | 0.65 | 0.66 | | Fuerte en POS, balance de clases aceptable, rendimiento estable |
| ***Naive Bayes - df_sub_mejorado*** | 0.75 | 0.74 | 0.92 | 0.57 | 0.73 | | Tiene buena precisión, pero un poco simple, menos robusto en la clase NEU |
| ***DeepSeek*** | 0.71 | 0.57 | 0.72 | 0.17 | 0.81 | 2.02s | Muy sesgado a NEG, mal desempeño en NEU |

**Conclusión:**

Después de haber entrenado y mejorado todos los modelos, se obtuvo un total de 18 entrenamientos, 4 entrenamientos por algoritmos tradicionales y dos para modelos LLMs. En todos los modelos tradicionales, el entrenamiento con datos balanceados y mejorados, fueron los que mejor se destacaron por modelado.

Se determino que los mejores modelos fueron:

1. ChatGpt.
2. SVM Mejorado con df_submuestreo.
3. XGBoost Mejorado con df_submuestreo.

Cabe mencionar que esta selección no se basó solo en las métricas en forma cuantitativa, sino que también se hizo una revisión en aspectos técnicos como una buena calibración probabilística o la robustez con respecto al desbalance entre las 3 clases y el comportamiento en las matrices de confusión de cada modelo.

En primer lugar, ChatGpt se llevó este puesto, gracias a su puntaje en el accuracy con el 86%, un F1-macro de 0.83 y también gracias a su buen desempeño con la precisión en las 3 clases. Cabe recordar que no son modelos entrenables debido a que su arquitectura se basa en modelos de lenguaje grandes (LLMs), lo que le permite tener una gran ventaja, gracias a que puede capturar bien la semántica y el contexto de cualquier tema especificado, que en este caso es el socialismo del siglo XXI. También su tiempo de inferencia por comentario fue bueno (0.67s).

Por otro lado, el segundo lugar fue para SVM, el cual se destacó entre los modelos tradicionales por su desempeño en F1-macro de 0.77 y en F1-score en POS fue bueno ya que

obtuvo un resultado de 0.93. Esto es útil ya que ayuda a captar percepciones positivas que haya en discursos políticos con respecto al socialismo. Además, los resultados de las otras clases no fueron malos. Como es un modelo basado en márgenes de separación optimas, si mostro un buen resultado en contextos donde se puedan presentar ambigüedad en comentarios que son muy polarizados.

Por último, XGBoost ocupo el tercer puesto, no por el resultado de sus métricas de por sí, sino por su robustez estructural, por tener una buena regularización interna y por obtener una calibración probabilística superior. Aunque en algunas métricas como Accuracy y F1POS fue superado por Random Forest, este modelo presentó un Log Loss bajo (0.6732). También obtuvo el tercer lugar debido a que las 3 clases fue proporcional y no hubo demasiado sobreajuste a las clases predominantes.

En general, estos 3 modelos representan una combinación bien equilibrada entre precisión y robustez. ChatGpt fue superior a los demás modelos por el hecho de ser un lenguaje LLM, SVM fue un buen modelo y ayudó a clasificar opiniones positivas, y XGBoost fue la opción que tiene más balance y ajustable donde se requiere una discriminación multiclase.

**Proceso de revisión**

Para la revisión de los modelos seleccionados se realizó el mismo proceso, es decir, se realizó el proceso de limpieza de datos, se envió los nuevos comentarios a los 3 modelos para revisar sus resultados.

Se hizo una nueva descarga de comentarios de YouTube de los siguientes videos:

***Tabla 36***

*Nuevos videos para prueba*

| ***Nombre*** | ***Video*** | ***Url*** |
|---|---|---|
| ¿ QUE ES EL SOCIALISMO DEL SIGLO XXI ? La mejor explicación | ***Figura70***<br>*Nuevo video de prueba 1*<br>  <br>*Nota: Recuperado de "Si no lo veo No lo creo"* | https://www.youtube.com/watch?v=wQhjC6Ui3eE |
| Socialismo del Siglo XXI: Origen y representantes | ***Figura71***<br>*Nuevo video de prueba 2*<br>  <br>*Nota: Recuperado de "El profe de historia"* | https://www.youtube.com/watch?v=SrSX2vXPaR0 |
| El socialismo del siglo 21 | ***Figura72***<br>*Nuevo video de prueba 3* | https://www.youtube.com/watch?v=r1j8QfXmQbA |


*Nota: Recuperado de "Alberto Montenegro"*

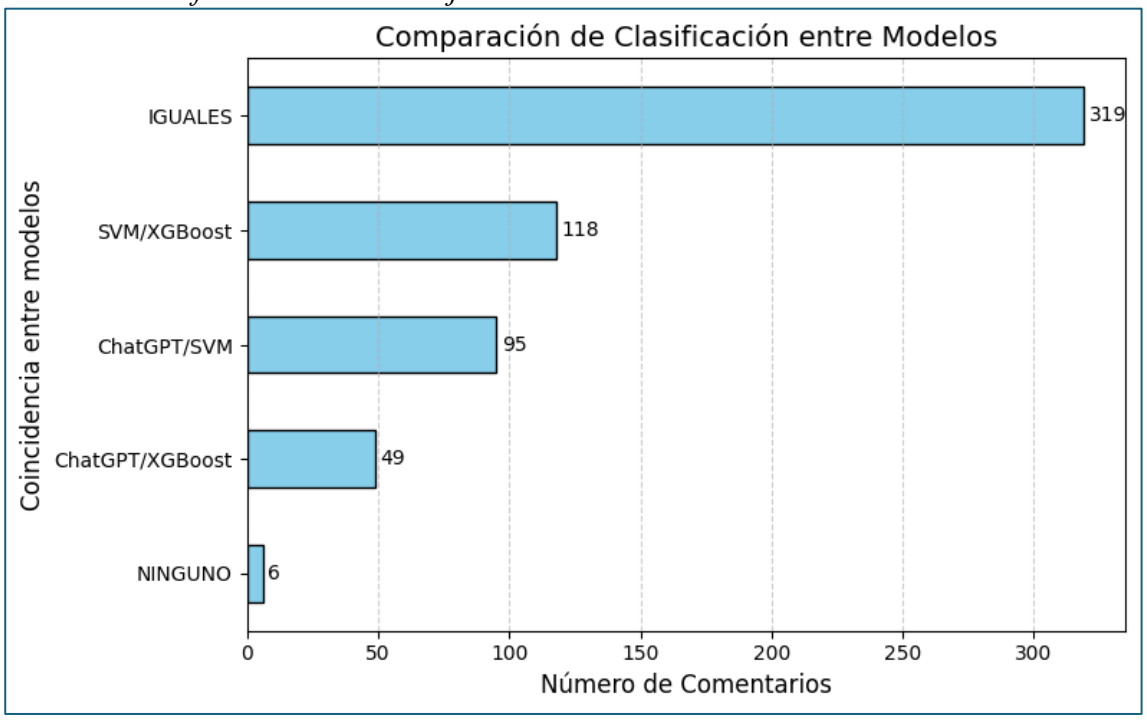
***Comparación de desempeño entre modelos clásicos y un modelo de IA (LLM)***

El total de comentarios que se obtuvo de estos 3 videos fue de 581 comentarios. A estos datos se les aplico la misma limpieza que se usó para el entrenamiento de datos, después de ello se tomó estos datos limpios y se los envió a los 3 mejores modelos donde ChatGpt, se demoró 4min 58,8s, SVM 0.5s, XGBoost 0,0s.

Finalmente se hizo una revisión de los resultados de cada modelo para ver sus discrepancias y coincidencias:

***Figura 73***

*Resultados de clasificación de los 3 mejores modelos*




Finalmente se pudo observar que hay un gran consenso entre los 3 modelos con un 54% de coincidencia entre sus resultados, esto indica un buen desempeño y estabilidad en los 3 modelos ya que represento más de la mitad de los comentarios (319 de 587). Sin embargo, también existe una coincidencia parcial entre dos modelos y uno que clasifico con distinta clase, la distribución de coincidencias entre dos clases fue la siguiente:

- **SVM/XGBoost:** 118 comentarios (20.1%)
- **ChatGPT/SVM:** 95 comentarios (16.2%)
- **ChatGPT/XGBoost:** 49 comentarios (8.3%)

Esto indicó que SVM y XGBoost, los modelos tradicionales, tienden a coincidir más lo que indica que ambos tienen un criterio de clasificación similar. Por otro lado, ChatGpt coincide más con SVM, lo que se pudo interpretar como SVM pudo capturar tanto patrones

lingüísticos como estructurales. Finalmente, se revisó que solo el 1% (6 comentarios) fueron clasificados con distintas clases en cada modelo, estos comentarios pueden tener ambigüedad o son difíciles de clasificarlos.

### *Determinar los siguientes pasos*

Se puede proceder con una implementación con casos reales, inclinándose en un enfoque hibrido, es decir, con los 3 modelos para mejorar la confianza en la clasificación.

Se recomienda optar por un enfoque donde se tenga una estrategia de cascada para clasificar los nuevos comentarios en la etapa de despliegue, donde, si los 3 modelos coinciden, se toma la predicción, si coinciden solo dos, se prioriza ese resultado y si no hay coincidencia, se toma el modelo más robusto.

Por otro lado, se puede iniciar un nuevo proyecto para refinar los casos donde los resultados son diferentes en los 3 modelos para mejorar el preprocesamiento o revisar y entrenar con estos comentarios ambiguos y complejos.

## Conclusiones y recomendaciones

### Conclusiones

Tras el análisis de más de 6mil comentarios extraídos desde la plataforma de YouTube, se pudo identificar que las percepciones hacia el socialismo en la última década han sido en su mayoría comentarios negativos. La etapa de visualización sirvió para confirmar las presencias de comentarios y frases que se lo asocian a la crítica, descontento y la gran polarización política que este tema tiene.

Se recopilo los comentarios de varios videos de YouTube que exponían su contenido político sobré el socialismo latinoamericano, provenientes de diversos canales y de diversos países. Estos comentarios pasaron por un proceso de limpieza antes de su entrenamiento posterior.

Se diseño, entreno y se evaluaron varios modelos de aprendizaje automático para la clasificación de comentarios por emociones agrupadas por, comentarios: positivos (POS), negativos (NEG), neutros (NEU). En total se entrenaron 4 modelos clásicos (Naive Bayes, Random Forest, SVM y XGBoost) con diferentes versiones, con la Data real (df_final) y con la data balanceada (df_submuestreo) y a cada uno se aplicó su respectiva mejora, Mientras que para los dos modelos de LLM (ChatGpt y DeepSeek) únicamente se los evaluó con el DataSet real (df_final). Todos los modelos tradicionales fueron mejorados con técnicas como GridSearch, validación cruzada y ajuste de hiperparámetros.

Se evaluó de forma detallada el rendimiento de cada entrenamiento del modelo de cada uno. Se saco métricas de evaluación típicas como, accuracy, precisión, recall, F1-score y adicionalmente se usó otras métricas de evaluación para algunos modelos como el índice Gini, log loss y rango de distancias en hiperplano.

Estas comparaciones ayudaron a revelar cual era el mejor modelo que se desempeñó mejor con estos datos, donde, ChatGpt se llevó el primero puesto, seguido de SVM y XGBoost. Lo cual dio a conocer la capacidad y robustez de este LLM para esta tarea, pero también no se quita méritos los modelos de aprendizaje tradicionales donde SVM fue el segundo mejor modelo en el ranking de evaluación. Esta comparativa también da a conocer que no por el

hecho de ser un modelo de tipo LLM es bueno, ya que se demostró que el modelo de DeepSeek fue el que peor desempeño mostro.

Después de haber evaluado los 6 modelos, se hizo una evaluación adicional con nuevos datos a los modelos que mejor desempeño mostraron (ChatGpt, SVM, XGBoost), donde se observó que entre los 3 modelos llegaron a un consenso en resultados del 54% que represento el 319 de comentarios de 587 que era el total de comentarios nuevos. Lo que refleja una buena estabilidad y confiabilidad en los resultados de los modelos. Alto a rescatar también fue que solo el 1%, que equivale a 6 comentarios de los datos nuevos, resultaron completamente ambiguos para los 3 modelos, es decir, los 3 modelos no coincidieron en los resultados.

**Recomendaciones**

Para afinar la precisión en la clasificación de estos comentarios se propone desarrollar un sistema de análisis de sentimientos, usando un enfoque hibrido, es decir usar los 3 modelos. Esto ayudará a aumentar la precisión y se tendrá una clasificación robusta del sistema.

Para tener una mejor clasificación, también se debe tomar en cuenta una lógica jerárquica, es decir, si los 3 modelos coinciden, se acepta el resultado predicho, si solo coinciden dos clases, se toma el resultado de estas dos clases, si ninguno de los modelos coincide, se usa el resultado predicho por ChatGpt que fue el mejor modelo.

Se puede trabajar a más profundidad en los comentarios donde no hubo coincidencias de resultados entre los 3 modelos, lo cual representa un punto crítico a revisión. Se recomienda crear un conjunto especializado con comentarios ambiguos o con sarcasmo para seguir entrenando estos modelos tradicionales y seguir mejorando su desempeño.

Este estudio puede ser de interés para investigadores, periodistas, sociólogos o politólogos los cuales buscan comprender las narrativas predominantes en los discursos ciudadanos sobre las posturas políticas que están presentes en la región.

Se recomienda considerar a futuro comentarios que incorporen más campos como país de origen, comentario por fecha, lo que podría ayudar a un análisis más detallado a nivel de tiempo, especio sobre las percepciones del socialismo.

## Bibliografía


1. History.com Editors. (2018, November 9). Socialism. History.com. https://www.history.com/topics/industrial-revolution/socialism
2. Liu, B. (2012). *Sentiment analysis and opinion mining*. Synthesis Lectures on Human Language Technologies, 5(1), 1–167. https://doi.org/10.2200/S00416ED1V01Y201204HLT016
3. Hamburger Fernández, Á. A. (2014). *El socialismo del siglo XXI en América Latina: características, desarrollos y desafíos*. Revista de Relaciones Internacionales, Estrategia y Seguridad, 9(1), 131–154.
4. Hagerty, A., & Rubinov, I. (2019). Global AI ethics: A review of the social impacts and ethical implications of artificial intelligence. AI & Society, 36, 55–72. https://arxiv.org/pdf/1907.07892
5. Vargas Chuquimia, A. L. (2017). Socialismo siglo XXI Latinoamérica [Manuscript]. Academia.edu. https://www.academia.edu/35268722/Socialismo_siglo_xxi_latinoamerica
6. *XXI: los casos de Cuba, Venezuela, Bolivia y México*. INEHRM.
7. Redalyc. (2007). *El socialismo del siglo XXI en América Latina*. *Revista de Estudios Latinoamericanos*.
8. Scielo Colombia. (2014). *El socialismo del siglo XXI en América Latina: características, desarrollos y desafíos*. *Revista CES*.
9. Scielo México. (2017). *América Latina en el siglo XXI: transiciones, malestares y retos*. *Revista Mexicana de Estudios Políticos y Sociales*.
10. Scielo Venezuela. (2009). *El pensamiento socialista latinoamericano y el desarrollo organizacional*. *Estudios Organizacionales*.

11. Scielo Chile. (2014). *Giro a la izquierda en la América Latina del siglo XXI*. *Revista de Ciencias Políticas*.
12. Duarte, J. M., & Berton, L. (2023). *A review of semi-supervised learning for text classification*. *Artificial Intelligence Review*, 56, 9401–9469.
13. Thangaraj, M., & Sivakami, M. (2018). *Text classification techniques: a literature review*. *International Journal of Information and Knowledge Management*, 13, 117–135.
14. Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3.ª ed.). Prentice Hall.
15. Mishra, T., Sutanto, E., Rossanti, R., Pant, N., Ashraf, A., Raut, A., Uwabareze, G., Oluwatomiwa, A., & Zeeshan, B. (2024). Use of large language models as artificial intelligence tools in academic research and publishing among global clinical researchers. Scientific Reports, 14(1), 31672. https://doi.org/10.1038/s41598-024-81370-6
16. Krotov, V., Johnson, L. R., & Silva, L. (2021). *Tutorial: Legality and ethics of web scraping*. Communications of the Association for Information Systems. https://www.researchgate.net/publication/352014123_Legality_and_Ethics_of_Web_Scraping
17. Harris, D. (2016, May 12). OkCupid study reveals the perils of big data science. Wired. https://www.wired.com/2016/05/okcupid-study-reveals-perils-big-data-science/
18. Iwasiński, Ł. (2021). *Social implications of algorithmic bias*. Uniwersytet Warszawski. https://www.researchgate.net/publication/349120634_SOCIAL_IMPLICATIONS_OF_ALGORITHMIC_BIAS
19. PLoS One. (2020). *Algorithmic bias in social research: A meta-analysis*. *PLOS One*.

20. Almuqati, M. T., Sidi, F., Mohd Rum, S. N., & Zolkepli, M. (2024). *Challenges in Supervised and Unsupervised Learning: A Comprehensive Overview*. *IJASEIT, 14*(4), 2088–5334.

21. Bharadiya, J. P. (2023). *A Comprehensive Survey of Deep Learning Techniques Natural Language Processing*. *European Journal of Technology, 7*(1), 58–66. https://doi.org/10.47672/ejt.1473

22. ResearchGate. (2023). *Understanding the Difference Between Supervised and Unsupervised Learning Techniques*.

23. Earl, R. (2024, Mayo 16). What is unsupervised machine learning? Lifewire. https://www.lifewire.com/what-is-unsupervised-learning-7555694

24. Earl, R. (2023, Junio 16). What is unsupervised machine learning? Lifewire. https://www.lifewire.com/what-is-supervised-learning-7508014

25. Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

26. Raschka, S. (2014, October 4). *Naive Bayes and text classification – Introduction and theory*. Sebastian Raschka Blog. https://sebastianraschka.com/Articles/2014_naive_bayes_1.html

27. Xu, S. (2016). Bayesian naïve Bayes classifiers to text classification. *Journal of Information Science*, 1–12. https://doi.org/10.1177/0165551510000000

28. Biau, G., & Scornet, E. (2016). A Random Forest Guided Tour. *Test*.

29. Jalal, N., Mehmood, A., & Choi, G. S. (2022). A semantics aware Random Forest for text classification. *Journal of King Saud University – Computer and Information Sciences, 34*(10), 2733–2742.

30. Tong, S., & Chang, E. Y. (2001). Support Vector Machine Active Learning with Applications to Text. *Journal of Machine Learning Research, 2*, 45–66.
31. Ukey, K. P., & Alvi, A. S. (2012). Text classification using support vector machine. *International Journal of Engineering Research & Technology (IJERT)*, 1(3), 1–4. https://es.scribd.com/document/873231324/Text-Classification-Using-Support-Vector-Machine-IJERTV1IS3174
32. Chen, M., Wu, Y., Wingerd, B., Liu, Z., Xu, J., Thakkar, S., Pedersen, T. J., Donnelly, T., Mann, N., Tong, W., Wolfinger, R. D., & Bao, W. (2024). Automatic text classification of drug-induced liver injury using document-term matrix and XGBoost. *Frontiers in Artificial Intelligence*, 7, Article 1401810. https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2024.1401810/full
33. Manning, C. D., Raghavan, P., & Schütze, H. (2009). *An introduction to information retrieval*. Cambridge University Press. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf
34. Aizawa, A. (2003). An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1), 45–65. https://ccc.inaoep.mx/~villasen/index_archivos/cursoTL/articulos/Aizawa-tf-idfMeasures.pdf
35. Piskorski, J., & Jacquet, G. (2020). TF-IDF character n-grams versus word embedding-based models for fine-grained event classification: A preliminary study. En A. Hürriyetoğlu, E. Yörük, V. Zavarella, & H. Tanev (Eds.), *Proceedings of the Workshop on Automated Extraction of Socio-political Events from News 2020* (pp. 26–34). European Language Resources Association (ELRA). https://aclanthology.org/2020.aespen-1.6.pdf

36. Sharma, A. (2024). *Large language models (LLMs): Architectures, applications, and future innovations in artificial intelligence*. https://www.researchgate.net/publication/388256187_Large_Language_Models_LLMs_Architectures_Applications_and_Future_Innovations_in_Artificial_Intelligence

37. Kalyan, K. S. (2023). *A survey of GPT-3 family large language models including ChatGPT and GPT-4*. Akmmus AI. https://www.researchgate.net/publication/374446998_A_Survey_of_GPT-3_Family_Large_Language_Models_Including_ChatGPT_and_GPT-4

38. Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). *Large Language Models: A Survey*. *IEEE Transactions on AI*.

39. Metz, C. E. (1978). Basic principles of ROC analysis. *Seminars in Nuclear Medicine*.

40. Abdel Aziz Taha, A. (2020). Metrics for evaluating 3D medical image segmentation. *BMC Medical Imaging*.

41. Friedman, J. H., et al. (2009). *The Elements of Statistical Learning*. Springer.

42. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

43. Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning, 20*(3), 273–297.

44. Moguerza, J. M., & Muñoz, M. A. (2006). Support vector machines in real applications. *Pattern Recognition*.

45. Olson, D. L., & Delen, D. (2008). *Advanced Data Mining Techniques*. Springer.

46. Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13*, 281–305.

47. Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation. *IJCAI*.

48. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.

49. Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

50. BM. (s. f.). **CRISP-DM Help Overview**. IBM SPSS Modeler Documentation.

51. IBM. (s. f.). **CRISP-DM in IBM SPSS Modeler**. IBM SPSS Modeler Documentation.

52. Data Science PM. (2024, diciembre 9). *What are the 6 CRISP-DM Phases?* DataScience-PM.com.

53. PSE. (2022). *CRISP-DM Methodology*. PSE Consultoria.

54. SV-Europe. (2022). *Crisp DM methodology*. Smart Vision Europe.

55. Data & Decision. (2023). *The key to Data Science success is the CRISP methodology*. Business & Decision.

56. Data Science PM. (2024). *What are the 6 CRISP-DM Phases?* DataScience-PM.com. https://www.datascience-pm.com/crisp-dm-2

57. IBM. (s. f.). *CRISP-DM Help Overview*. IBM SPSS Modeler Documentation. https://www.ibm.com/docs/en/spss-modeler

58. IBM. (s. f.). *CRISP-DM Methodology in SPSS Modeler*. IBM Knowledge Center. https://www.ibm.com/docs/en/spss-modeler

59. Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 2, 1137–1145.

60. PSE Consultoría. (2022). *CRISP-DM Methodology*. https://www.pse.pe/crisp-dm/

61. Smart Vision Europe. (2022). *CRISP-DM: The Industry Standard Data Mining Process Model*. https://www.sv-europe.com/crisp-dm-methodology/

62. Business & Decision. (2023). *The key to Data Science success is the CRISP methodology*. https://www.businessdecision.com

## Anexos

1. **Función para descargar los comentarios de YouTube:**

```
from googleapiclient.discovery import build
import pandas as pd

# Configuración de la API
API_KEY = "AIzaSyC2tcK_gRa"
youtube = build("youtube", "v3", developerKey=API_KEY)

# Lista de IDs de videos
video_ids = [
    "83LCyPXnbVA", "1LwQCiwue3g", "npn8r7wh9Ko", "62tda2CkQ8s",
"k14KVEBCx50",
    "y5VmGaOUojM", "nC-AK7bhUfc", "TjpfJGGuwTc", "A8nq_Q2_KBY",
"HZa5voSUlS0",
    "g3OtscDTzNc", "yP51AWdcmLI", "CHsGEOw2PlY", "l8XU2_2B298",
"yxsSVkEopEs",
    "48p81I_We_8", "VgSJ7JS720c"
]

def get_comments(video_id):
    comments_data = []
    request = youtube.commentThreads().list(
        part="snippet",
        videoId=video_id,
        maxResults=100,
        textFormat="plainText"
    )
    while request:
        response = request.execute()
        for item in response["items"]:
            snippet = item["snippet"]["topLevelComment"]["snippet"]
            comments_data.append({
                "video_id": video_id,
```

```
            "author": snippet.get("authorDisplayName"),
            "text": snippet.get("textDisplay"),
            "published_at": snippet.get("publishedAt"),
            "like_count": snippet.get("likeCount")
        })
    request = youtube.commentThreads().list_next(request, response)
  return comments_data

# Recolectar comentarios de todos los videos
all_comments = []
for vid in video_ids:
  print(f"Descargando comentarios de video: {vid}")
  comments = get_comments(vid)
  all_comments.extend(comments)

# Convertir a DataFrame
df = pd.DataFrame(all_comments)

# Guardar en CSV
output_file = "youtube_comments.csv"
df.to_csv(output_file, index=False, encoding="utf-8")
print(f"Se han guardado {len(df)} comentarios en '{output_file}'")
```

2. **Función para eliminar StopWords y emojis para visualización:**

```
nltk.download('stopwords')
stopwords_es = set(stopwords.words('spanish'))

# Agregar stopwords personalizadas comunes en comentarios
stopwords_personalizadas = {
  'q', 'si', 'solo', 'ser', 'mas', 'bien', 'año', 'años', 'puede', 'pueden', 'tener',
  'tener', 'hacer', 'está', 'están', 'estoy', 'somos', 'soy', 'va', 'van', 'vez', 'vezes',
  'aqui', 'ahi', 'asi', 'pues', 'otro', 'otra', 'eso', 'este', 'ese', 'muy', 'más', 'menos'
}
stopwords_es.update(stopwords_personalizadas)

# Función para quitar tildes
def quitar_tildes(texto):
  return ''.join(
    c for c in unicodedata.normalize('NFD', texto)
    if unicodedata.category(c) != 'Mn'
```

```
    )

# Función final de limpieza completa
def limpiar_texto(texto):
    texto = texto.lower()
    texto = quitar_tildes(texto)
    texto = texto.lower()
    texto = re.sub(r"http\S+|www\S+|https\S+", '', texto)
    texto = re.sub(r'\@\w+|\#', '', texto)
    texto = re.sub(r'[^a-záéíóúñü\s]', '', texto)
    texto = re.sub(r'\s+', ' ', texto).strip()
    texto = emoji.replace_emoji(texto, replace='')
    texto = re.sub(r'["“”‘’\'\[\],.:;!?¿¡()]', '', texto)
    texto = re.sub(r'[^a-zñü\s]', '', texto)
    palabras = texto.split()
    palabras_filtradas = [p for p in palabras if p not in stopwords_es]
    return ' '.join(palabras_filtradas)
```

### 3. Función para llamar a la API de OPEN AI.

```
# === CLAVE OPENAI ===
client = OpenAI(api_key=" ")

# === FUNCIÓN GPT PARA CLASIFICAR N-GRAMAS ===
def clasificar_con_gpt(texto):
    try:
        prompt = f"Clasifica el sentimiento del siguiente comentario como POS
(positivo), NEG (negativo) o NEU (neutral). Solo responde POS, NEG o
NEU:\n\n'{texto}'"

        respuesta = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": "Eres un experto en análisis de
sentimientos."},
                {"role": "user", "content": prompt}
            ],
            temperature=0
        )
        salida = respuesta.choices[0].message.content.strip().upper()
        return salida if salida in ["POS", "NEG", "NEU"] else "NEU"
    except Exception as e:
```

```
        print(f" Error con: {texto[:50]}... -> {e}")
        return "ERROR"

# === EXTRACCIÓN DE BIGRAMAS Y TRIGRAMAS ===
vectorizer = CountVectorizer(ngram_range=(2,3), max_features=50)
X = vectorizer.fit_transform(df['texto_limpio'])  # Asegúrate que 'df' contiene esa
columna

n_gramas = vectorizer.get_feature_names_out()
frecuencias = X.sum(axis=0).A1

df_ngramas = pd.DataFrame({'comentario': n_gramas, 'frecuencia': frecuencias})
df_ngramas = df_ngramas.sort_values(by='frecuencia', ascending=False)

# Etiqueta si es BIGRAMA o TRIGRAMA
df_ngramas['ngrama'] = df_ngramas['comentario'].apply(lambda x: 'BIGRAMA' if
len(x.split()) == 2 else 'TRIGRAMA')

# Reordenar columnas
df_ngramas = df_ngramas[['ngrama', 'comentario']]

# === CLASIFICAR CADA N-GRAMA CON BARRA DE PROGRESO ===
tqdm.pandas(desc=" Clasificando n-gramas con ChatGPT")
df_ngramas['sentimiento'] =
df_ngramas['comentario'].progress_apply(clasificar_con_gpt)

# === GUARDAR A CSV ===
df_ngramas.to_csv("ngramas_clasificados_chatgpt.csv", index=False)
print(" Clasificación completada y guardada en 'ngramas_clasificados.csv'!")
```

4. **Función para clasificar con Hugging Face**

```
# Cargar modelo de Hugging Face
modelo = "nlptown/bert-base-multilingual-uncased-sentiment"

clasificador = pipeline("sentiment-analysis", model=modelo, tokenizer=modelo)

# Función de clasificación
def clasificar_sentimiento(texto):
    resultado = clasificador(texto[:512])[0]  # Limitar a 512 tokens
    label = resultado['label']
    if label in ['1 star', '2 stars']:
```

```
        return 'NEG'
    elif label == '3 stars':
        return 'NEU'
    else:
        return 'POS'

# Aplicar al DataFrame
df['sentimiento'] = df['texto_modelado'].apply(clasificar_sentimiento)
```

### 5. Modelo Beto-Sentiment-analysis

```
# Cargar modelo entrenado en español
modelo_esp = "finiteautomata/beto-sentiment-analysis"
clasificador_esp = pipeline("sentiment-analysis", model=modelo_esp,
tokenizer=modelo_esp)
# Función para clasificar con el modelo en español
def clasificar_sentimiento_es(texto):
    resultado = clasificador_esp(texto[:512])[0]
    label = resultado['label']
    # Ya devuelve POS / NEG / NEU directamente
    return label

# Aplicar la función y guardar resultados en una nueva columna
df['sentimiento_esp'] = df['texto_modelado'].apply(clasificar_sentimiento_es)
```

### 6. Modelo Naive Bayes df_final

```
# === Función para entrenar y evaluar el modelo con NB ===
def entrenar_y_evaluar_nb(X_train, y_train, X_test, y_test,
nombre_modelo='Naive Bayes', cmap_color='Blues'):
    pipeline = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('nb', MultinomialNB())
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    # Clasificasión
    print(f" Evaluación del modelo {nombre_modelo}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # Matriz de confusión
```

```
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre_modelo}',
['NEG', 'NEU', 'POS'], cmap='Blues')
    plt.tight_layout()
    plt.show()

    return pipeline  # se  guarda el modelo
#-------------------------------------------------#
#LLAMADA
modelo_nb_final = entrenar_y_evaluar_nb(X_final_train, y_final_train,
X_final_test, y_final_test,
                                        nombre_modelo='Naive Bayes - df_final',
                                        cmap_color='Blues')
```

### 7. Modelo Naive Bayes df_final mejorado

```
# === Función para optimizar Naive Bayes ===
def optimizar_y_evaluar_nb(X_train, y_train, X_test, y_test, nombre='NB
Mejorado - df_final'):
    # Pipeline
    pipeline = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('nb', MultinomialNB())
    ])

    # GridSearch y los hiperparámetros modificados
    param_grid = {
        'tfidf__ngram_range': [(1, 1), (1, 2)],
        'tfidf__min_df': [1, 2, 5],
        'tfidf__max_df': [0.9, 0.95, 1.0],
        'nb__alpha': [0.1, 0.5, 1.0]
    }

    # Grid Search
    grid = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1,
scoring='f1_macro', verbose=1)
    grid.fit(X_train, y_train)
    best_model = grid.best_estimator_
    y_pred = best_model.predict(X_test)

    # Evaluación
    print(" Mejor combinación de hiperparámetros:")
```

```
    print(grid.best_params_)
    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # Matriz de confusión
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Greens')
    plt.tight_layout()
    plt.show()

    return best_model
#-------------------------------------------------#
modelo_nb_mejorado = optimizar_y_evaluar_nb(X_final_train, y_final_train,
X_final_test, y_final_test)
```

### 8. Modelo Naive Bayes df_submuestreo

```
def optimizar_y_evaluar_nb_sub(X_train, y_train, X_test, y_test, nombre='NB
df_submuestreo'):

    # === Pipeline ===
    pipeline = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('nb', MultinomialNB())
    ])

    # === GridSearch con los hiperparámetros ===
    param_grid = {
        'tfidf__ngram_range': [(1, 1), (1, 2)],
        'tfidf__min_df': [1, 2, 5],
        'tfidf__max_df': [0.9, 0.95, 1.0],
        'nb__alpha': [0.1, 0.5, 1.0]
    }

    # === Grid Search ===
    grid = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1,
scoring='f1_macro', verbose=1)
    grid.fit(X_train, y_train)
    best_model = grid.best_estimator_
    y_pred = best_model.predict(X_test)
```

```
    # === Evaluación ===
    print(" Mejor combinación de hiperparámetros:")
    print(grid.best_params_)
    print(f"\nEvaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Matriz de Confusión ===
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Blues')
    plt.tight_layout()
    plt.show()

    return best_model
#-------------------------------------------------#
modelo_nb_sub_mejorado = optimizar_y_evaluar_nb_sub(X_sub_train,
y_sub_train, X_sub_test, y_sub_test)
```

### 9. Modelo Naive Bayes df_submuestreo mejorado

```
def optimizar_y_evaluar_nb_count(df_label='NB Mejorado - df_submuestreo',
                                 X_train=None, y_train=None,
                                 X_test=None, y_test=None):
    # === Pipeline ===
    pipeline = Pipeline([
        ('vect', CountVectorizer()),
        ('nb', MultinomialNB(fit_prior=False))
    ])

    # === GridSearch y sus hiperparámetros ===
    param_grid = {
        'vect__ngram_range': [(1, 1), (1, 2)],
        'vect__min_df': [1, 2],
        'vect__max_df': [0.9, 0.95],
        'nb__alpha': [0.1, 0.5, 1.0]
    }

    # === GridSearch ===
    grid = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1,
                        scoring='f1_macro', verbose=1, error_score='raise')
```

```
    grid.fit(X_train, y_train)
    best_model = grid.best_estimator_
    y_pred = best_model.predict(X_test)

    # === Resultados ===
    print(" Mejores hiperparámetros:")
    print(grid.best_params_)
    print(f"\n Evaluación del modelo {df_label}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Matriz de Confusión ===
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {df_label}', ['NEG',
'NEU', 'POS'], cmap='Greens')
    plt.tight_layout()
    plt.show()

    return best_model
# ------------------------------------------------------#

modelo_nb_count_mejorado = optimizar_y_evaluar_nb_count(
    df_label='Naive Bayes - df_submuestreo mejorado',
    X_train=X_sub_train, y_train=y_sub_train,
    X_test=X_sub_test, y_test=y_sub_test
)
```

**10. Modelo Random Forest df_final**

```
def entrenar_y_evaluar_rf(X_train, y_train, X_test, y_test, nombre='Random
Forest - df_final'):


    # === Vectorización TF-IDF ===
    vectorizer = TfidfVectorizer()
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # === Entrenamiento modelo ===
    rf_model = RandomForestClassifier(random_state=42)
    rf_model.fit(X_train_vec, y_train)
```

```
    # === Predicción y evaluación ===
    y_pred = rf_model.predict(X_test_vec)
    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Matriz de Confusión ===
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Blues')
    plt.tight_layout()
    plt.show()
    from sklearn.metrics import roc_auc_score
    from sklearn.preprocessing import label_binarize

    # Binarizamos las clases (NEG=0, NEU=1, POS=2)
    y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
    y_score_proba = rf_model.predict_proba(X_test_vec)
    # Calcular AUC (promedio macro)
    auc = roc_auc_score(y_test_bin, y_score_proba, average='macro',
multi_class='ovr')

    # Índice Gini
    gini_index = 2 * auc - 1
    print(f" Índice de Gini (macro promedio): {gini_index:.4f}")
    return rf_model, vectorizer
#-------------------------------------------------#
modelo_rf_final, vectorizer_rf_final = entrenar_y_evaluar_rf(
    X_train=X_final_train, y_train=y_final_train,
    X_test=X_final_test, y_test=y_final_test,
    nombre='Random Forest - df_final'
)
```

### 11. Modelo Random Forest df_final mejorado

```
def optimizar_y_evaluar_rf_final(X_train, y_train, X_test, y_test,
nombre='Random Forest Optimizado - df_final'):

    # === Vectorizador ajustado ===
    vectorizer = TfidfVectorizer(
        max_df=0.95,
        min_df=5,
```

```
        ngram_range=(1, 2),
        sublinear_tf=True
    )

    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # === Hiperparámetros para búsqueda ===
    param_grid = {
        'n_estimators': [100, 200],
        'max_depth': [20, None],
        'min_samples_split': [2, 5],
        'max_features': ['sqrt', 'log2'],
        'class_weight': ['balanced']
    }

    # === GridSearch ===
    grid = GridSearchCV(
        RandomForestClassifier(random_state=42),
        param_grid=param_grid,
        cv=3,
        scoring='f1_macro',
        n_jobs=-1,
        verbose=2
    )
    grid.fit(X_train_vec, y_train)
    best_model = grid.best_estimator_

    print(" Mejores hiperparámetros:")
    print(grid.best_params_)

    # === Predicción y métricas ===
    y_pred = best_model.predict(X_test_vec)
    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Índice de Gini ===
    y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
    y_proba = best_model.predict_proba(X_test_vec)
    auc = roc_auc_score(y_test_bin, y_proba, average='macro', multi_class='ovr')
    gini_index = 2 * auc - 1
    print(f" Índice de Gini (macro promedio): {gini_index:.4f}")

    # === Matriz de Confusión ===
    cm = confusion_matrix(y_test, y_pred)
```

```
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Greens')
    plt.tight_layout()
    plt.show()

    return best_model, vectorizer
#-------------------------------------------------#
modelo_rf_final_opt, vectorizer_rf_final_opt = optimizar_y_evaluar_rf_final(
    X_train=X_final_train, y_train=y_final_train,
    X_test=X_final_test, y_test=y_final_test
)
```

### 12. Modelo Random Forest df_submuestreo

```
def optimizar_y_evaluar_rf_sub(X_train, y_train, X_test, y_test,
nombre='Random Forest - df_submuestreo'):

    # === Vectorizador optimizado ===
    vectorizer = TfidfVectorizer(
        max_df=0.95,
        min_df=5,
        ngram_range=(1, 2),
        sublinear_tf=True
    )

    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # === Grid de hiperparámetros ===
    param_grid = {
        'n_estimators': [100, 200],
        'max_depth': [20, None],
        'min_samples_split': [2, 5],
        'max_features': ['sqrt', 'log2'],
        'class_weight': ['balanced']
    }

    # === Búsqueda de mejores parámetros ===
    grid = GridSearchCV(
        RandomForestClassifier(random_state=42),
        param_grid=param_grid,
        cv=3,
```

```
        scoring='f1_macro',
        n_jobs=-1,
        verbose=2
    )

    grid.fit(X_train_vec, y_train)
    best_model = grid.best_estimator_

    print(" Mejores hiperparámetros:")
    print(grid.best_params_)

    # === Predicción y métricas ===
    y_pred = best_model.predict(X_test_vec)
    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Índice de Gini ===
    y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
    y_proba = best_model.predict_proba(X_test_vec)
    auc = roc_auc_score(y_test_bin, y_proba, average='macro', multi_class='ovr')
    gini_index = 2 * auc - 1
    print(f" Índice de Gini (macro promedio): {gini_index:.4f}")

    # === Matriz de Confusión ===
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Blues')
    plt.tight_layout()
    plt.show()

    return best_model, vectorizer

#-------------------------------------------------#
modelo_rf_sub_opt, vectorizer_rf_sub_opt = optimizar_y_evaluar_rf_sub(
    X_train=X_sub_train, y_train=y_sub_train,
    X_test=X_sub_test, y_test=y_sub_test
)
```

**13. Modelo Random Forest df_submuestreo mejorado**

```
def optimizar_y_evaluar_rf_sub_mejorado(X_train, y_train, X_test, y_test,
nombre='Random Forest Mejorado - df_submuestreo'):
```

```
    # === Vectorizador mejorado ===
    vectorizer = TfidfVectorizer(
        max_df=0.95,
        min_df=3,
        ngram_range=(1, 2),
        sublinear_tf=True,
        max_features=10000
    )

    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # === Grid de hiperparámetros mejorado ===
    param_grid = {
        'n_estimators': [200, 300, 500],
        'max_depth': [30, 50, None],
        'min_samples_split': [2, 5],
        'min_samples_leaf': [1, 3, 5],
        'max_features': ['sqrt', 'log2'],
        'class_weight': ['balanced']
    }

    grid = GridSearchCV(
        RandomForestClassifier(random_state=42),
        param_grid=param_grid,
        cv=3,
        scoring='f1_macro',
        n_jobs=-1,
        verbose=2
    )

    grid.fit(X_train_vec, y_train)
    best_model = grid.best_estimator_

    print(" Mejores hiperparámetros:")
    print(grid.best_params_)

    # === Predicción y métricas ===
    y_pred = best_model.predict(X_test_vec)
    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Índice de Gini ===
    y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
    y_proba = best_model.predict_proba(X_test_vec)
```

```
    auc = roc_auc_score(y_test_bin, y_proba, average='macro', multi_class='ovr')
    gini_index = 2 * auc - 1
    print(f" Índice de Gini (macro promedio): {gini_index:.4f}")

    # === Matriz de Confusión ===
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Greens')
    plt.tight_layout()
    plt.show()

    return best_model, vectorizer

#-------------------------------------------------#
modelo_rf_sub_opt, vectorizer_rf_sub_opt =
optimizar_y_evaluar_rf_sub_mejorado(
    X_train=X_sub_train, y_train=y_sub_train,
    X_test=X_sub_test, y_test=y_sub_test
)
```

### 14. Modelo SVM df_final

```
def entrenar_y_evaluar_svm_final(X_train, y_train, X_test, y_test,
nombre='SVM - df_final'):
    # === Vectorizador TF-IDF ===
    vectorizer = TfidfVectorizer(
        max_df=0.95,
        min_df=5,
        ngram_range=(1, 2),
        sublinear_tf=True
    )
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # === Modelo SVM ===
    svm_model = LinearSVC(random_state=42)
    svm_model.fit(X_train_vec, y_train)
    y_pred = svm_model.predict(X_test_vec)

    # === Evaluación ===
    print(f"\n Evaluación del modelo {nombre}:")
```

```
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Coeficientes de decisión (distancia al hiperplano)
    decision_scores = svm_model.decision_function(X_test_vec)
    print(f"\nRango de distancias al hiperplano (coeficiente de decisión):")
    print(f"↪ Mínimo: {np.min(decision_scores):.4f} | Máximo:
{np.max(decision_scores):.4f}")

    # === Matriz de Confusión ===
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Blues')
    plt.tight_layout()
    plt.show()

    return svm_model, vectorizer
#-------------------------------------------------#
modelo_svm_final, vectorizer_svm_final = entrenar_y_evaluar_svm_final(
    X_train=X_final_train, y_train=y_final_train,
    X_test=X_final_test, y_test=y_final_test
)
```

### 15. Modelo SVM df_final mejorado

```
def optimizar_y_evaluar_svm_final(X_train, y_train, X_test, y_test,
nombre='SVM Optimizado - df_final'):
    # === Vectorización avanzada ===
    vectorizer = TfidfVectorizer(
        max_df=0.95,
        min_df=5,
        ngram_range=(1, 2),
        sublinear_tf=True
    )
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # === Grid de hiperparámetros ===
    param_grid = [
        {
            'C': [0.1, 1, 10],
            'kernel': ['linear'],
```

```
            'class_weight': [None, 'balanced']
        },
        {
            'C': [0.1, 1, 10],
            'kernel': ['rbf'],
            'class_weight': [None, 'balanced'],
            'gamma': ['scale', 'auto']
        }
    ]

    # === GridSearchCV ===
    grid = GridSearchCV(
        SVC(random_state=42),
        param_grid=param_grid,
        cv=3,
        scoring='f1_macro',
        n_jobs=-1,
        verbose=2
    )

    grid.fit(X_train_vec, y_train)
    best_model = grid.best_estimator_

    print("Mejores hiperparámetros:")
    print(grid.best_params_)

    # === Predicción y métricas ===
    y_pred = best_model.predict(X_test_vec)
    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Coeficientes de decisión (distancia al hiperplano)
    try:
        decision_scores = best_model.decision_function(X_test_vec)
        print(f"\n Rango de distancias al hiperplano (coeficiente de decisión):")
        print(f"↪  Mínimo: {np.min(decision_scores):.4f} | Máximo:
{np.max(decision_scores):.4f}")
    except Exception as e:
        print(" Este modelo no soporta decision_function:", e)

    # === Matriz de Confusión ===
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Greens')
```

```
    plt.tight_layout()
    plt.show()

    return best_model, vectorizer

#-----------------------------------------------#
modelo_svm_final_opt, vectorizer_svm_final_opt =
optimizar_y_evaluar_svm_final(
    X_train=X_final_train, y_train=y_final_train,
    X_test=X_final_test, y_test=y_final_test
)
```

### 16. Modelo SVM df_submuestreo

```
def entrenar_y_evaluar_svm_sub(X_train, y_train, X_test, y_test, nombre='SVM
- df_submuestreo'):

    # === Vectorización TF-IDF (igual que antes) ===
    vectorizer = TfidfVectorizer(
        max_df=0.95,
        min_df=5,
        ngram_range=(1, 2),
        sublinear_tf=True
    )
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # === Entrenamiento con LinearSVC ===
    svm_model = LinearSVC(random_state=42)
    svm_model.fit(X_train_vec, y_train)
    y_pred = svm_model.predict(X_test_vec)

    # === Evaluación
    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Coeficiente de decisión
    decision_scores = svm_model.decision_function(X_test_vec)
    print(f"\n Rango de distancias al hiperplano:")
    print(f"↪  Mínimo: {np.min(decision_scores):.4f} | Máximo:
{np.max(decision_scores):.4f}")
```

```
    # === Matriz de Confusión
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Blues')
    plt.tight_layout()
    plt.show()

    return svm_model, vectorizer
#-----------------------------------------------#
modelo_svm_sub, vectorizer_svm_sub = entrenar_y_evaluar_svm_sub(
    X_train=X_sub_train, y_train=y_sub_train,
    X_test=X_sub_test, y_test=y_sub_test
)
```

### 17. Modelo SVM df_submuestreo mejorado

```
def optimizar_y_evaluar_svm_sub_mejorado(X_train, y_train, X_test, y_test,
nombre='SVM Mejorado - df_submuestreo'):

    # === Vectorización TF-IDF mejorada ===
    vectorizer = TfidfVectorizer(
        max_df=0.95,
        min_df=3,                  # más tolerancia a palabras no comunes
        ngram_range=(1, 2),
        sublinear_tf=True,
        max_features=15000         # más características
    )
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # === GridSearch ampliado ===
    param_grid_svm = [
        {
            'C': [0.01, 0.1, 1, 5, 10, 50],
            'kernel': ['linear'],
            'class_weight': ['balanced']
        },
        {
            'C': [0.01, 0.1, 1, 5, 10],
            'kernel': ['rbf'],
            'gamma': ['scale', 'auto', 0.01, 0.001],
            'class_weight': ['balanced']
```

```
        }
    ]

    # === GridSearchCV ===
    grid = GridSearchCV(
        SVC(random_state=42),
        param_grid=param_grid_svm,
        cv=3,
        scoring='f1_macro',
        n_jobs=-1,
        verbose=2
    )

    grid.fit(X_train_vec, y_train)
    best_model = grid.best_estimator_

    print(" Mejores hiperparámetros encontrados:")
    print(grid.best_params_)

    # === Predicciones
    y_pred = best_model.predict(X_test_vec)
    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # === Coeficientes de decisión
    try:
        decision_scores = best_model.decision_function(X_test_vec)
        print(f"\n Rango de distancias al hiperplano:")
        print(f"↪  Mínimo: {np.min(decision_scores):.4f} | Máximo:
{np.max(decision_scores):.4f}")
    except Exception as e:
        print(" Este modelo no soporta decision_function:", e)

    # === Matriz de Confusión
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Greens')
    plt.tight_layout()
    plt.show()

    return best_model, vectorizer
#-----------------------------------------------#
modelo_svm_sub_opt, vectorizer_svm_sub_opt =
optimizar_y_evaluar_svm_sub_mejorado(
```

```
    X_train=X_sub_train, y_train=y_sub_train,
    X_test=X_sub_test, y_test=y_sub_test
)
```

### 18. Modelo XGBoost df_final

```
# === Función principal para entrenar y evaluar con XGBoost usando log loss ===
def entrenar_y_evaluar_xgb(X_train, y_train, X_test, y_test, nombre='XGBoost - df_final'):
    # === Vectorización TF-IDF ===
    vectorizer = TfidfVectorizer()
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # === Entrenamiento del modelo ===
    xgb_model = XGBClassifier(
        use_label_encoder=False,
        eval_metric='mlogloss',
        objective='multi:softprob',
        random_state=42
    )
    xgb_model.fit(X_train_vec, y_train)

    # === Predicción y evaluación ===
    y_pred = xgb_model.predict(X_test_vec)
    y_proba = xgb_model.predict_proba(X_test_vec)

    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU', 'POS']))


    # === Cálculo de Log Loss ===
    loss = log_loss(y_test, y_proba, labels=[0, 1, 2])
    print(f" Log Loss (entropía cruzada promedio): {loss:.4f}")

    # === Matriz de Confusión ===
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG', 'NEU', 'POS'], cmap='Blues')
    plt.tight_layout()
```

```
    plt.show()


    return xgb_model, vectorizer
#-----------------------------------------------#
modelo_xgb_final, vectorizer_xgb_final = entrenar_y_evaluar_xgb(
    X_train=X_final_train,
    y_train=y_final_train,
    X_test=X_final_test,
    y_test=y_final_test,
    nombre='XGBoost - df_final'
)
```

### 19. Modelo XGBoost df_final mejorado

```
# === Función para dibujar la matriz de confusión ===
def draw_confusion_matrix(cm, ax, title, labels, cmap="Greens"):
    sns.heatmap(cm, annot=True, fmt='d', cmap=cmap,
                xticklabels=labels, yticklabels=labels, ax=ax)
    ax.set_xlabel("Predicción")
    ax.set_ylabel("Valor real")
    ax.set_title(title)

# === Entrenamiento y optimización con XGBoost - df_final ===
def entrenar_y_optimizar_xgb_final(X_train, y_train, X_test, y_test,
nombre='XGBoost Mejorado - df_final'):
    # Vectorización TF-IDF
    vectorizer = TfidfVectorizer()
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # Espacio de búsqueda de hiperparámetros
    param_dist = {
        'max_depth': randint(3, 8),
        'learning_rate': uniform(0.01, 0.3),
        'n_estimators': randint(100, 300),
        'subsample': uniform(0.8, 0.2),
        'colsample_bytree': uniform(0.8, 0.2)
    }

    # Modelo base
    base_model = XGBClassifier(
        objective='multi:softprob',
```

```
        eval_metric='mlogloss',
        use_label_encoder=False,
        random_state=42,
        verbosity=0
    )

    print(" Iniciando búsqueda de hiperparámetros (RandomizedSearchCV, 30
combinaciones)...\n")

    random_search = RandomizedSearchCV(
        estimator=base_model,
        param_distributions=param_dist,
        n_iter=30,
        scoring='neg_log_loss',
        cv=3,
        verbose=1,
        n_jobs=-1,
        random_state=42
    )

    random_search.fit(X_train_vec, y_train)
    best_model = random_search.best_estimator_

    print(f"\n Mejores hiperparámetros encontrados:
{random_search.best_params_}")

    # Evaluación
    y_pred = best_model.predict(X_test_vec)
    y_proba = best_model.predict_proba(X_test_vec)

    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Greens')
    plt.tight_layout()
    plt.show()

    loss = log_loss(y_test, y_proba, labels=[0, 1, 2])
    print(f" Log Loss (entropía cruzada promedio): {loss:.4f}")

    return best_model, vectorizer
#-----------------------------------------------#
```

```
modelo_xgb_final_mejorado, vectorizer_xgb_final_mejorado =
entrenar_y_optimizar_xgb_final(
    X_train=X_final_train,
    y_train=y_final_train,
    X_test=X_final_test,
    y_test=y_final_test,
    nombre='XGBoost Mejorado - df_final'
)
```

### 20. Modelo XGBoost df_submuestreo

```
# === Entrenamiento y evaluación XGBoost ===
def entrenar_y_evaluar_xgb(X_train, y_train, X_test, y_test, nombre='XGBoost -
df_submuestreo'):
    # Vectorización
    vectorizer = TfidfVectorizer()
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # Modelo
    xgb_model = XGBClassifier(
        objective='multi:softprob',
        eval_metric='mlogloss',
        use_label_encoder=False,
        random_state=42
    )
    xgb_model.fit(X_train_vec, y_train)

    # Predicciones
    y_pred = xgb_model.predict(X_test_vec)
    y_proba = xgb_model.predict_proba(X_test_vec)

    # Reporte
    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    # Log Loss
    loss = log_loss(y_test, y_proba, labels=[0, 1, 2])
    print(f" Log Loss (entropía cruzada promedio): {loss:.4f}")
    # Matriz de confusión
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
```

```
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Blues')
    plt.tight_layout()
    plt.show()



    return xgb_model, vectorizer
#-----------------------------------------------#
modelo_xgb_sub, vectorizer_xgb_sub = entrenar_y_evaluar_xgb(
    X_train=X_sub_train,
    y_train=y_sub_train,
    X_test=X_sub_test,
    y_test=y_sub_test,
    nombre='XGBoost - df_submuestreo'
)
```

**21. Modelo XGBoost df_submuestreo mejorado**

```
# === Entrenamiento y optimización rápida con XGBoost ===
def entrenar_y_optimizar_xgb_rapido(X_train, y_train, X_test, y_test,
nombre='XGBoost Mejorado - df_submuestreo'):
    # Vectorización TF-IDF
    vectorizer = TfidfVectorizer()
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # Espacio de búsqueda reducido
    param_dist = {
        'max_depth': randint(3, 8),
        'learning_rate': uniform(0.01, 0.3),
        'n_estimators': randint(100, 300),
        'subsample': uniform(0.8, 0.2),
        'colsample_bytree': uniform(0.8, 0.2)
    }

    # Modelo base
    base_model = XGBClassifier(
        objective='multi:softprob',
        eval_metric='mlogloss',
        use_label_encoder=False,
        random_state=42,
        verbosity=0
```

```
    )

    print(" Iniciando búsqueda de hiperparámetros (RandomizedSearchCV, 15
combinaciones)...\n")

    random_search = RandomizedSearchCV(
        estimator=base_model,
        param_distributions=param_dist,
        n_iter=30,
        scoring='neg_log_loss',
        cv=3,
        verbose=1,
        n_jobs=-1,
        random_state=42
    )

    random_search.fit(X_train_vec, y_train)
    best_model = random_search.best_estimator_

    print(f"\n Mejores hiperparámetros encontrados:
{random_search.best_params_}")

    # Evaluación
    y_pred = best_model.predict(X_test_vec)
    y_proba = best_model.predict_proba(X_test_vec)

    print(f"\n Evaluación del modelo {nombre}:")
    print(classification_report(y_test, y_pred, target_names=['NEG', 'NEU',
'POS']))

    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(6, 5))
    draw_confusion_matrix(cm, ax, f'Matriz de Confusión - {nombre}', ['NEG',
'NEU', 'POS'], cmap='Greens')
    plt.tight_layout()
    plt.show()

    loss = log_loss(y_test, y_proba, labels=[0, 1, 2])
    print(f" Log Loss (entropía cruzada promedio): {loss:.4f}")

    return best_model, vectorizer
#-----------------------------------------------#
modelo_xgb_sub_mejorado, vectorizer_xgb_sub_mejorado =
entrenar_y_optimizar_xgb_rapido(
    X_train=X_sub_train,
    y_train=y_sub_train,
```

```
    X_test=X_sub_test,
    y_test=y_sub_test,
    nombre='XGBoost Mejorado - df_submuestreo'
                                        )
```

### 22. Modelo ChatGpt

```
# ApiKey cliente OpenAI
client = OpenAI(api_key="sk-proj-")
# Función para clasificar un texto usando ChatGPT
def clasificar_con_gpt(texto):
    try:
        prompt = f"Clasifica el sentimiento del siguiente comentario como POS (positivo), NEG (negativo) o NEU (neutral). Solo devuelve POS, NEG o NEU:\n\n'{texto}'"

        respuesta = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": "Eres un experto en análisis de sentimientos."},
                {"role": "user", "content": prompt}
            ],
            temperature=0
        )
        salida = respuesta.choices[0].message.content.strip().upper()
        return salida if salida in ["POS", "NEG", "NEU"] else "NEU"  # fallback
    except Exception as e:
        print(f" Error con comentario: {texto[:50]}... -> {e}")
        return "ERROR"

# === DataFrame base ===
df_base = df_final.copy()

# Crear una copia con las columnas necesarias
df_nuevo = df_base[['texto_modelado', 'sentimiento_final', 'sentimiento_encoded']].copy()

# Barra de progreso
tqdm.pandas(desc="Clasificando con ChatGPT")
df_nuevo['sentimiento_gpt'] =
df_nuevo['texto_modelado'].progress_apply(clasificar_con_gpt)
```

```
# Guarda el resultado
df_nuevo.to_csv("comentarios_clasificados_chatgpt.csv", index=False)
```

### 23. Modelo DeepSeek

```
from concurrent.futures import ThreadPoolExecutor, as_completed
import pandas as pd
import httpx
import time
from tqdm import tqdm

# === Configuración API DeepSeek ===
API_KEY = "sk-"  # tu key
ENDPOINT = "https://api.deepseek.com/v1/chat/completions"

headers = {
    "Authorization": f"Bearer {API_KEY}",
    "Content-Type": "application/json"
}

# === Función para clasificar usando DeepSeek ===
def clasificar_deepseek(texto):
    prompt = f"Clasifica el sentimiento del siguiente comentario como POS (positivo), NEG (negativo) o NEU (neutral). Solo responde POS, NEG o NEU:\n\n{texto}"
    data = {
        "model": "deepseek-coder",
        "messages": [
            {"role": "system", "content": "Eres un experto en análisis de sentimientos."},
            {"role": "user", "content": prompt}
        ],
        "temperature": 0
    }

    for intento in range(3):
        try:
            response = httpx.post(ENDPOINT, headers=headers, json=data, timeout=60)
            response.raise_for_status()
            result = response.json()
            salida = result["choices"][0]["message"]["content"].strip().upper()
            return salida if salida in ["POS", "NEG", "NEU"] else "NEU"
        except Exception as e:
            print(f" Intento {intento+1} falló para: {texto[:50]}... -> {e}")
            time.sleep(2)
```

```
    return "ERROR"

# === DataFrame base ===
df_base_deep = df_final[['texto_modelado', 'sentimiento_final',
'sentimiento_encoded']].copy()

# === Procesamiento paralelo con barra de progreso
resultados = [None] * len(df_base_deep)  # Reservar espacio

with ThreadPoolExecutor(max_workers=5) as executor:
    futuros = {executor.submit(clasificar_deepseek, texto): idx for idx, texto in
enumerate(df_base_deep['texto_modelado'])}
    for future in tqdm(as_completed(futuros), total=len(futuros), desc="Clasificando
con DeepSeek"):
        idx = futuros[future]
        try:
            resultados[idx] = future.result()
        except Exception as e:
            print(f"Error en índice {idx} -> {e}")
            resultados[idx] = "ERROR"

# === Asignar resultados y guardar
df_base_deep["sentimiento_deepseek"] = resultados
df_base_deep.to_csv("comentarios_clasificados_deepseek.csv", index=False)
```

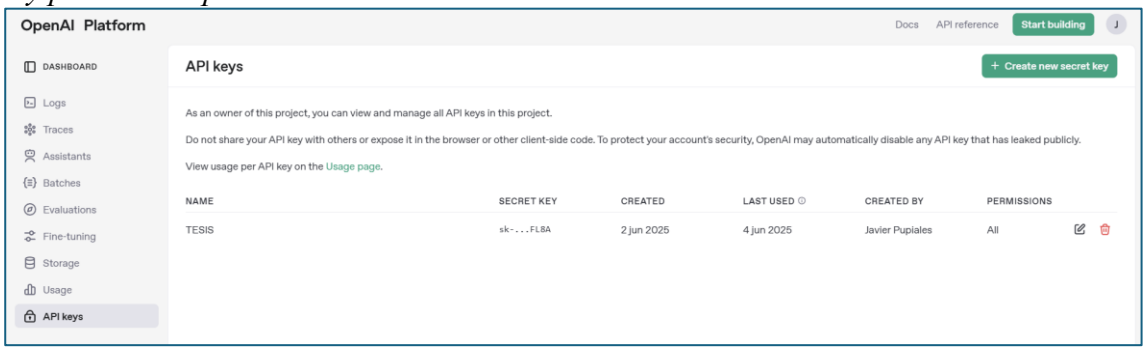
### 24. Precios API ChatGpt

**API Key:**

Para sacar la API de ChatGpt, se debe tener una cuenta creada, a partir de ahí se procedió a generar una APIKey en el gestor de plataforma de OpenAI, en este apartado se puede crear varias APIkeys por proyectos. Una vez se crea la APIKey, se debe agregar un saldo en la billetera de billing, el cual el saldo mínimo que se puede recargar es 10 dólares.

***Figura 74***

*ApiKey para ChatGpt*



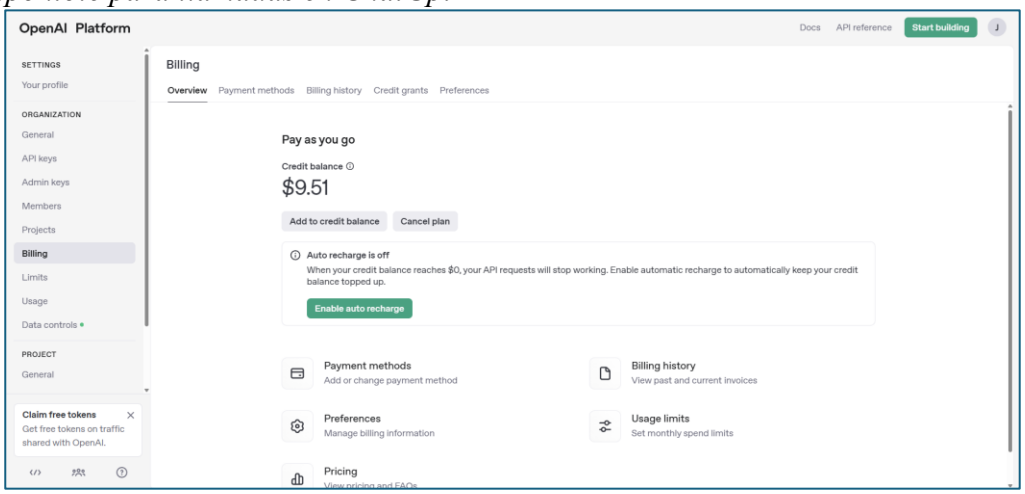

*Nota: Obtenido de "https://platform.openai.com/settings/organization/api-keys"*

**Saldo:**

A continuación, se muestra el saldo disponible que se va reduciendo a medida que se hace consumo de la API.

***Figura 75***

*Saldo disponible para llamadas en ChatGpt*

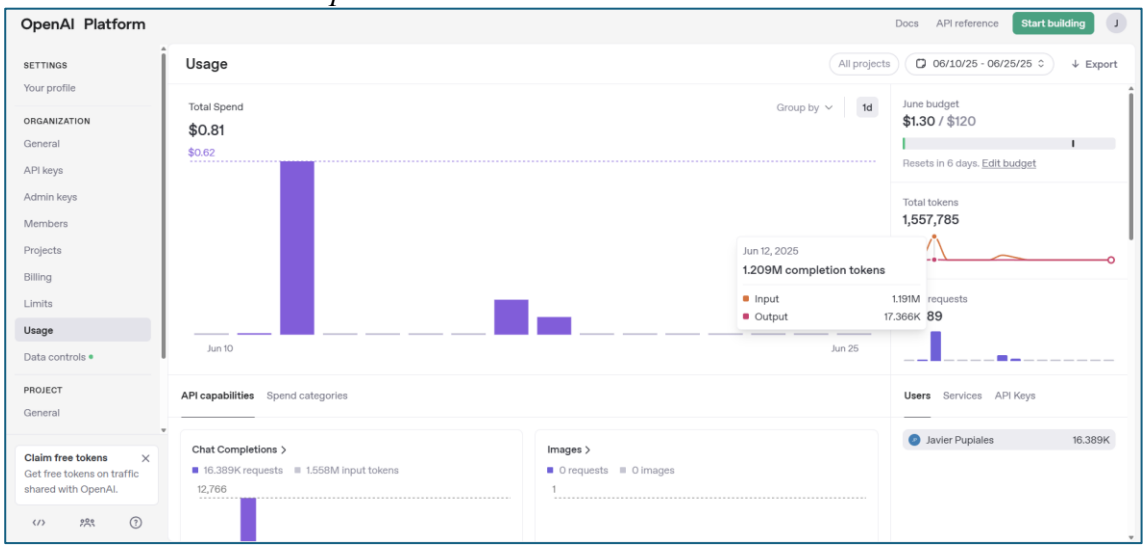



Nota: Recuperado de "https://platform.openai.com/settings/organization/billing/overview"

En la pestaña de uso se puede visualizar estadísticas del uso de la API, como, número de solicitudes enviadas, el número de tokens consumidos, el monto consumido actualmente, los días que se ha hecho llamadas a la API y demás.

***Figura 76***

*Estadística de uso API ChatGpt*



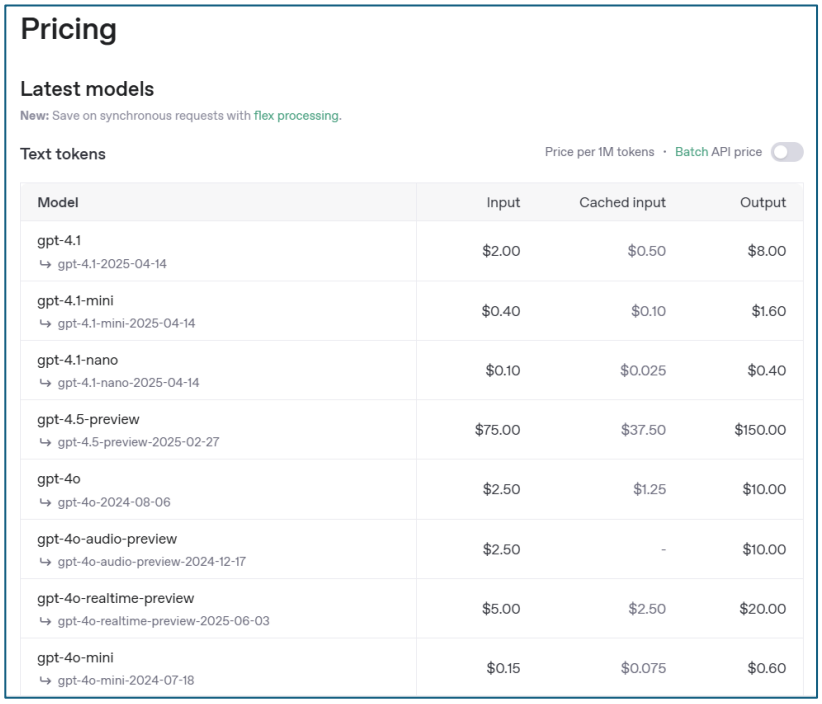

*Nota: Obtenido de "https://platform.openai.com/settings/organization/usage"*

**Precios:**

A continuación, se muestran los precios del consumo de la API por Token y por modelo. Cabe mencionar que ChatGpt tiene una amplia variedad de APIs que se puede utilizar para tareas cotidianas, tareas complejas, ejecución de salidas con BAtch API, API para generación de imágenes y más.

***Figura 77***

*Precios para el consumo de API ChatGpt*



Pricing

Latest models

**New:** Save on synchronous requests with flex processing.

Text tokens — Price per 1M tokens · Batch API price

| Model | Input | Cached input | Output |
|---|---|---|---|
| gpt-4.1<br>↳ gpt-4.1-2025-04-14 | $2.00 | $0.50 | $8.00 |
| gpt-4.1-mini<br>↳ gpt-4.1-mini-2025-04-14 | $0.40 | $0.10 | $1.60 |
| gpt-4.1-nano<br>↳ gpt-4.1-nano-2025-04-14 | $0.10 | $0.025 | $0.40 |
| gpt-4.5-preview<br>↳ gpt-4.5-preview-2025-02-27 | $75.00 | $37.50 | $150.00 |
| gpt-4o<br>↳ gpt-4o-2024-08-06 | $2.50 | $1.25 | $10.00 |
| gpt-4o-audio-preview<br>↳ gpt-4o-audio-preview-2024-12-17 | $2.50 | - | $10.00 |
| gpt-4o-realtime-preview<br>↳ gpt-4o-realtime-preview-2025-06-03 | $5.00 | $2.50 | $20.00 |
| gpt-4o-mini<br>↳ gpt-4o-mini-2024-07-18 | $0.15 | $0.075 | $0.60 |

*Nota: Recuperado de "https://platform.openai.com/docs/pricing"*

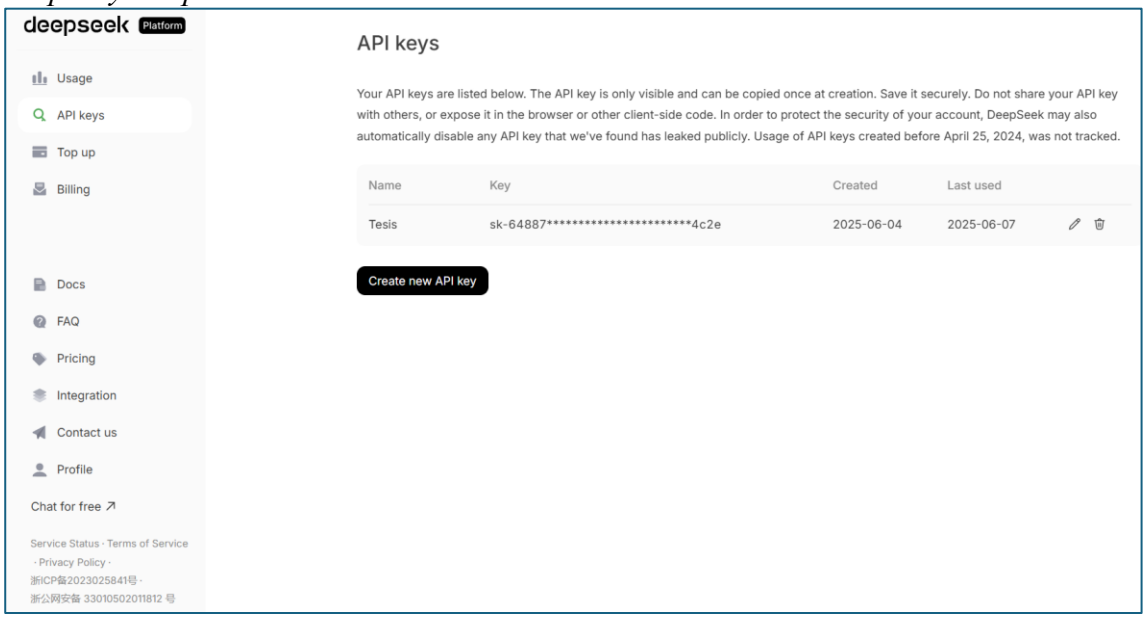
## 25. Precios API DeepSeek

### API Key:

El proceso de consumir una API para DeepSeek es lo mismo, para ello, se debe tener una cuenta registrada en la plataforma, se debe generar una APIKey con la que se va a consumir. Después de ello, se debe cargar salgo en la billing donde el precio mínimo para cargar es de 2 dólares.

***Figura 78***

*Clave ApiKey DeepSeek*



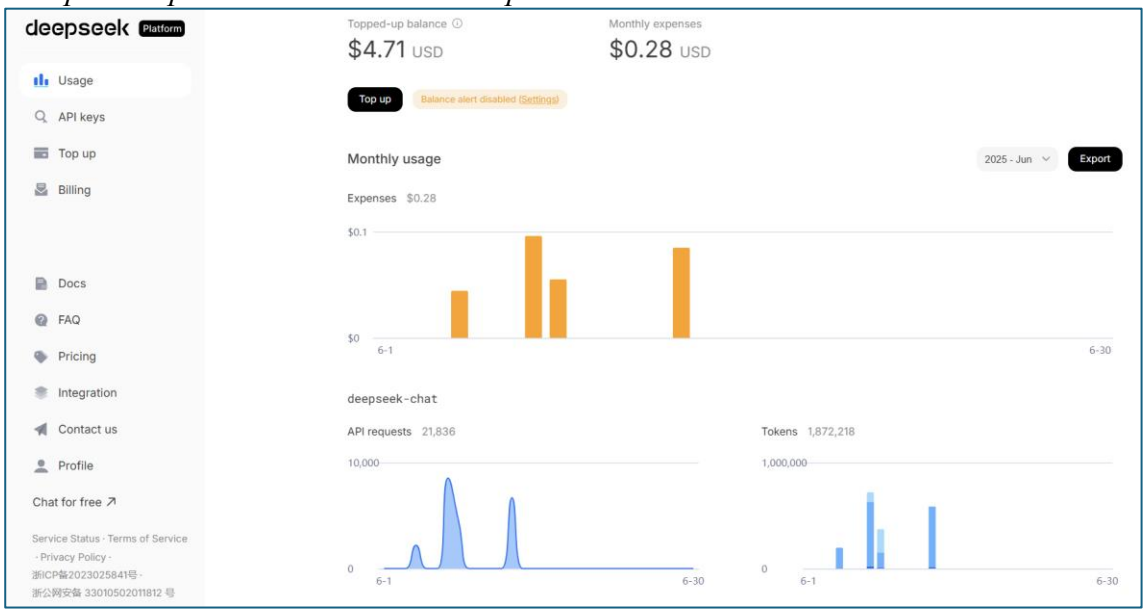

*Nota: Recuperado de "https://platform.deepseek.com/api_keys"*

**Saldo:**

El saldo consumido, se va reduciendo a medida que se hace consumo en la API, se muestra los días que se ha consumido, el valor que se consume, la cantidad de peticiones enviadas, la cantidad de tokens utilizados.

***Figura 79***

*Saldo disponible para consumo de API DeepSeek*




*Nota: Recuperado de "https://platform.deepseek.com/usage"*

**Precios:**

A continuación, se muestran los costos de la API por Token y por Modelo:

***Figura 80***

*Precios para consumo de API DeepSeek*

| MODELO (1) | | deepseek-chat | deepseek-razonador |
|---|---|---|---|
| LONGITUD DEL CONTEXTO | | 64K | 64K (2) |
| SALIDA MÁXIMA (3) | | PREDETERMINADO: 4K<br>MÁXIMO: 8K | PREDETERMINADO: 32K<br>MÁXIMO: 64K |
| CARACTERÍSTICAS | Salida Json | ✓ | ✓ |
| | Función Llamadas | ✓ | ✓ |
| | Prefijo de Chat Finalización (Beta) | ✓ | ✓ |
| | FIM Completion (Beta) | ✓ | ✗ |
| PRECIO ESTÁNDAR (UTC 00:30-16:30) | ENTRADA DE TOKENS 1M (HIT DE CACHÉ) (4) | $0.07 | $0.14 |
| | ENTRADA DE TOKENS 1M (CACHE MISS) | $0.27 | $0.55 |
| | SALIDA DE TOKENS 1M (5) | $1.10 | $2.19 |
| PRECIO DE DESCUENTO (6) (UTC 16:30-00:30) | ENTRADA DE TOKENS 1M (HIT DE CACHÉ) | $0.035 (50% OFF) | $0.035 (75% OFF) |
| | ENTRADA DE TOKENS 1M (CACHE MISS) | $0.135 (50% OFF) | $0.135 (75% OFF) |
| | SALIDA DE TOKENS 1M | $0.550 (50% OFF) | $0.550 (75% OFF) |

*Nota: recuperado de https://api-docs.deepseek.com/quick_start/pricing/*