COMILLAS
UNIVERSIDAD PONTIFICIA

Facultad de Ciencias Económicas y Empresariales
ICADE

LLM PARA LA DETECCIÓN DE NOTICIAS FALSAS: LA EVOLUCIÓN DE LOS MODELOS

Autor: Clara Oquiñena Goyena
Director: Susana Josefa Gago Rodríguez

MADRID | Abril 2026

INDICE

INDICE DE FIGURAS

RESUMEN

En la actualidad, la sociedad se encuentra expuesta a un flujo constante de información digital, caracterizado por la limitada presencia de mecanismos de filtrado y la democratización en la generación de contenidos. Este contexto ha favorecido la proliferación de noticias falsas, las cuales constituyen una amenaza para la estabilidad social. Asimismo, la inmediatez de las redes sociales y el consumo fragmentado de información contribuyen a amplificar este fenómeno, que se ve agravado por la escasa verificación de las fuentes y la falta de atención crítica por parte de los usuarios, facilitando así la difusión de la desinformación.

En este marco, el presente Trabajo de Fin de Grado analiza la evolución de las soluciones tecnológicas orientadas a combatir la infodemia, evaluando la eficacia de los Modelos de Lenguaje de Gran Tamaño (LLM) en comparación con enfoques tradicionales de aprendizaje automático. El objetivo principal de la investigación es determinar si la transición desde métodos basados en reglas y características superficiales hacia modelos capaces de capturar el contexto semántico y el razonamiento lingüístico representa una mejora significativa en la detección de noticias falsas.

Desde una perspectiva experimental, el estudio emplea la base de datos "*The Spanish Fake News Corpus"* y realiza un Análisis Exploratorio de Datos (EDA) con el fin de identificar patrones lingüísticos característicos de la desinformación, como la simplicidad estructural y una elevada carga emocional, especialmente asociada a la ira.

En cuanto a la metodología, se lleva a cabo una comparativa entre modelos clásicos de aprendizaje automático (como Random Forest, SVM y XGBoost) y el modelo de última generación Falcon3-1B. Los resultados obtenidos evidencian un avance significativo, reflejado tanto en la reducción de la necesidad de procesamiento manual como en la obtención de métricas elevadas, alcanzando una precisión del 90 % y un recall del 95 % en la clase de noticias falsas tras la optimización del umbral de clasificación.

En conclusión, los modelos de lenguaje de gran tamaño se consolidan como herramientas eficaces para mitigar los efectos de la desinformación. Su capacidad para comprender el lenguaje en profundidad, adaptarse a contextos dinámicos y su continuo desarrollo sugieren un escenario prometedor en la construcción de una sociedad más crítica, informada y resiliente frente a los desafíos informativos actuales.

Palabras clave: Detección de noticias falsas, Desinformación, Procesamiento del lenguaje natural, Modelos de lenguaje de gran tamaño

SUMMARY

Today, society is exposed to a constant flow of digital information, characterised by a lack of filtering mechanisms and the democratisation of content creation. This context has encouraged the proliferation of fake news, which poses a threat to social stability. Furthermore, the immediacy of social media and the fragmented consumption of information contribute to amplifying this phenomenon, which is exacerbated by the lack of source verification and critical engagement on the part of users, thereby facilitating the spread of misinformation.

Within this framework, this Final Year Project analyses the evolution of technological solutions aimed at combating the infodemic, evaluating the effectiveness of Large Language Models (LLMs) in comparison with traditional machine learning approaches. The main objective of the research is to determine whether the transition from rule-based methods and superficial features to models capable of capturing semantic context and linguistic reasoning represents a significant improvement in the detection of fake news.

From an experimental perspective, the study uses the "Spanish Fake News Corpus" database and conducts Exploratory Data Analysis (EDA) to identify linguistic patterns characteristic of disinformation, such as structural simplicity and a high emotional charge, particularly associated with anger.

In terms of methodology, a comparison is carried out between classical machine learning models (such as Random Forest, SVM and XGBoost) and the state-of-the-art Falcon3-1B model. The results obtained demonstrate significant progress, reflected both in the reduced need for manual processing and in the achievement of high metrics, reaching an accuracy of 90% and a recall of 95% in the fake news class following the optimisation of the classification threshold.

In conclusion, large language models are establishing themselves as effective tools for mitigating the effects of disinformation. Their ability to understand language in depth, adapt to dynamic contexts and their ongoing development suggest a promising future in building a society that is more critical, informed and resilient in the face of current information challenges.

Keywords: Detection of fake news, Disinformation, Natural language processing, Large language models

1. Introducción

En el presente Trabajo de Fin de Grado se analizará el funcionamiento de las noticias falsas, los patrones que siguen y se implementarán diversos modelos de clasificación con el objetivo de evaluarlos. Asimismo, se estudiará no solo el grado de desarrollo y mejora de estos modelos para su detección, sino también la eficacia de los modelos de lenguaje de gran tamaño (LLM) en esta tarea específica.

1.1 Motivación

La motivación para la realización de este Trabajo de Fin de Grado nace de una observación directa de mi entorno cotidiano y de cómo la tecnología está transformando nuestra percepción de la realidad. Como joven nativa digital, paso gran parte de mi tiempo expuesta a un flujo constante de información en redes sociales y plataformas digitales cuya procedencia es, a menudo, incierta. En mi día a día, he podido comprobar cómo, tanto en mi entorno cercano como a nivel personal, en ocasiones se aceptan ideas impactantes o noticias sorprendentes sin llegar a contrastarlas, especialmente en un contexto en el que la evolución de la Inteligencia Artificial y el desarrollo de vídeos generados mediante esta tecnología dificultan cada vez más la distinción entre información verídica y contenido manipulado.

Un ejemplo personal que marcó mi interés por este tema fue una noticia que leí sobre una cadena de restauración que solía frecuentar. La información (de origen dudoso, como se comprobó posteriormente) afirmaba que dicho establecimiento realizaba prácticas ilegales con sus productos. Tras compartir esta noticia con mi entorno, dejamos de acudir a dicho local hasta que finalmente se confirmó que el contenido era falso. Este hecho me llevó a reflexionar sobre la magnitud del problema: cuántas personas podrían haber tomado decisiones similares y cuál podría haber sido el impacto real de esta desinformación en la reputación y actividad de una empresa inocente.

Esta vivencia personal refleja, en realidad, un desafío de carácter global. La sociedad actual se caracteriza por una sobreexposición informativa y por una creciente inmediatez, lo que favorece que los usuarios acepten o rechacen información en función de sus creencias previas en lugar de contrastarla con fuentes fiables. En el ámbito empresarial, este fenómeno representa un riesgo significativo, ya que la difusión de noticias falsas puede afectar gravemente a la reputación corporativa, erosionar la confianza del público e incluso provocar variaciones en el valor de mercado en periodos muy reducidos de

tiempo, dado que tanto consumidores como inversores tienden a reaccionar de forma cada vez más impulsiva.

Fue precisamente a partir de esta reflexión cuando identifiqué la relación directa con los conocimientos adquiridos durante el grado en Business Analytics. Al observar cómo una única publicación puede influir en el comportamiento del consumidor y en la percepción de una marca, surgió mi interés por la aplicación del análisis de datos y de modelos predictivos como herramienta para abordar este problema. En este sentido, este trabajo parte del objetivo de transformar una inquietud personal en una aproximación técnica, explorando cómo el desarrollo de modelos de detección de veracidad puede contribuir a la lucha contra la desinformación y a la protección de los agentes implicados en el entorno digital.

1.2 Objetivos

El **objetivo principal** de este Trabajo de Fin de Grado es analizar cómo los avances recientes en el procesamiento del lenguaje natural (NLP), que han dado lugar a los modelos de lenguaje a gran escala (LLM), están contribuyendo al desarrollo de enfoques más eficaces para la detección de noticias falsas.

El TFG cuenta ademas con una serie de **objetivos secundarios**, en los que se busca, entre otros, comparar el rendimiento de los modelos basados en embeddings frente a los LLM (como Falcon3-1B) para determinar si verdaderamente hay ganancia en precisión y robustez. Ademas de ellos queremos también detectar y clasificar noticias como verdaderas o falsas mediante el uso de modelos de lenguaje (LLM) y variables contextuales, con el fin de prevenir las consecuencias negativas de la desinformación que tanto afectan a la sociedad de hoy en día. Y analizar las características comunes de las noticias falsas, tanto a nivel semántico (como el tono o sentimiento) como contextual, para identificar patrones que permitan su detección automática.

1.3 Metodología

Este Trabajo de Fin de Grado se desarrolla mediante una metodología mixta, que combina elementos cualitativos (descriptivos) y cuantitativos. La parte descriptiva se emplea para contextualizar el fenómeno de la desinformación digital y explicar los fundamentos de los modelos de lenguaje y técnicas de aprendizaje automático utilizadas. El análisis

cuantitativo, por su parte, permite medir la relación entre las características de las noticias y su veracidad, así como evaluar el rendimiento del modelo propuesto.
El enfoque metodológico adoptado es predominantemente inductivo, ya que se parte de análisis de datos reales para extraer patrones generales que permitan identificar noticias falsas. No obstante, también se incorpora un componente deductivo, al formular hipótesis sobre la influencia de ciertas variables (como la reputación de la fuente o el tono del texto) en la veracidad de las noticias, que serán contrastadas empíricamente.

1.4 Estructura

En este Trabajo de Fin de Grado, en primer lugar, se ha desarrollado un marco teórico en el que se analizó la evolución de los métodos de comunicación, con el objetivo de comprender cómo se ha llegado a la situación actual, en la que la capacidad de distinguir entre noticias falsas y verdaderas resulta especialmente relevante. A continuación, se examinó en profundidad el concepto de desinformación, su alcance y sus implicaciones. Finalmente, dentro de este marco teórico, se abordó el procesamiento del lenguaje natural (NLP) utilizado para analizar noticias, así como el funcionamiento y las características de los modelos de lenguaje a gran escala (LLM).

Una vez establecido este contexto teórico, se pasó a la fase de investigación práctica. Se utilizó una base de datos de noticias con el fin de identificar diferencias relevantes entre noticias falsas y verdaderas en cuanto a fuentes, temáticas, sentimientos, emociones y estilos de escritura. Posteriormente, se implementaron modelos de detección de noticias falsas basados en embeddings y, a continuación, modelos basados en LLM, con el propósito de evaluar su aplicabilidad en la actualidad y comparar su rendimiento para determinar cuáles ofrecen los mejores resultados.

Finalmente, se analizaron las principales diferencias entre los LLM y los modelos basados en embeddings, así como las mejoras recientes en estos enfoques y las posibles líneas de investigación futura.

2. Marco Teórico

2.1 Evolución de los métodos de comunicación

La difusión de la información ha evolucionado significativamente a lo largo de la historia. Según indica Menendez (2025), inicialmente, el control del conocimiento estaba concentrado **en un reducido grupo de actores**, como escribas, sacerdotes y el Estado, quienes decidían qué información era accesible para la sociedad. Con la invención de la imprenta y el surgimiento de la prensa escrita, la capacidad de difundir información se amplió considerablemente. Posteriormente, la radio y la televisión diversificaron aún más el ecosistema mediático, disminuyendo la exclusividad informativa de la prensa.

En la actualidad, la difusión de la información no se limita a medios tradicionales: gran parte de los contenidos se transmiten a través de redes sociales, blogs, influencers, creadores de contenido y cualquier persona con acceso a Internet, independientemente de su identidad o reputación. La tecnología ha introducido además nuevas formas de generación y difusión de información, incluyendo la inteligencia artificial (IA), que agiliza la producción de noticias y contenidos multimedia, aunque puede comprometer su fiabilidad, precisión y transparencia.

Con tan solo disponer de **conexión a internet**, es posible acceder de manera continua a toda esta gran diversidad de noticias e información. Según datos del Banco Mundial (2024), el 71 % de la población mundial utiliza Internet. En España, el Instituto Nacional de Estadística (2024) indica que, en 2024, el 96,8 % de los hogares contaba con acceso a Internet, mientras que en la Unión Europea el porcentaje alcanzaba el 94,1 %. Este crecimiento exponencial, como se puede visualizar en la figura 1, ha propiciado numerosos avances en poco tiempo, aunque también ha generado un vacío regulatorio y, en algunos casos, desconocimiento sobre su uso adecuado.

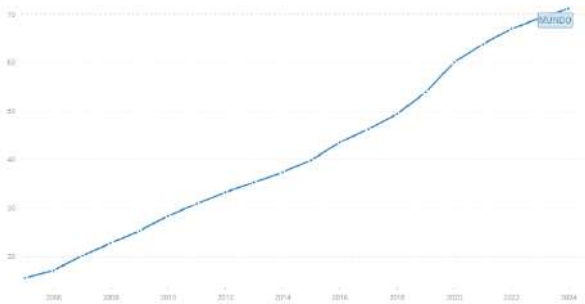


Figura 1: Individuos que utilizan Internet (% de la población)

Fuente: Banco Mundial (2025)

De hecho, un estudio realizado por Newman (2025) revela **un declive de los medios tradicionales** (televisión, prensa escrita y sitios web periodísticos) **mientras que redes sociales, plataformas de vídeo y agregadores de contenido han ganado protagonismo como fuentes informativas**. Influencers y creadores de contenido se consolidan como referentes, especialmente entre los jóvenes. Lo que es más, el consumo de información en estos medios es fragmentado: Facebook, YouTube, Instagram, TikTok y WhatsApp concentran solo el 10 % del alcance semanal y, además, han cambiado las preferencias de formato: el 55 % de la población prefiere ver noticias, el 31 % leerlas y el 14 % escucharlas.

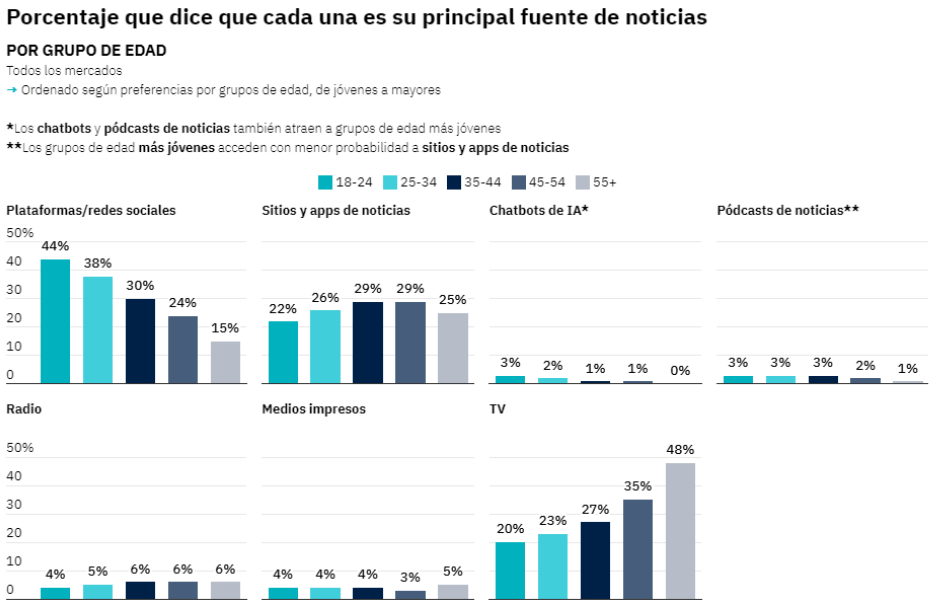


Figura 2: Fuentes principales de noticias

Fuente: Newman (2025)

Estos cambios generan preocupaciones sociales. **El 58 % de los encuestados manifestó temor a no poder distinguir si las noticias son verdaderas o falsas**. Esto se explica por la expansión del acceso a Internet y por el hecho de que cualquier persona, aunque no esté cualificada ni tenga experiencia periodística, puede difundir información, especialmente mediante dichas plataformas como redes sociales que están adquiriendo cada vez más relevancia como fuentes de información, aumentando asi la circulación de contenido falso o polarizado.

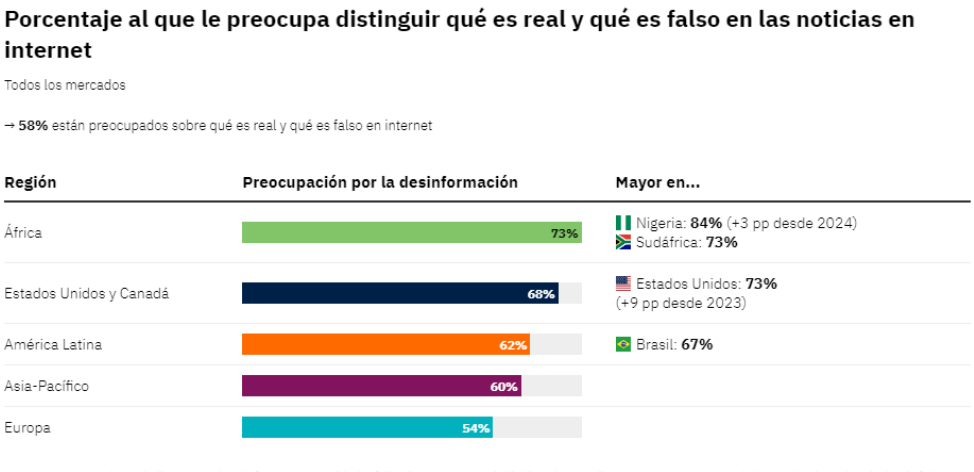


Figura 3: Preocupación por la veracidad de las noticias

Fuente: Newman (2025)

Aplicaciones como Facebook, TikTok, X, Instagram y YouTube son percibidas como las principales plataformas donde se difunden noticias falsas, mientras que políticos e influencers son los actores que más frecuentemente las divulgan. Esta situación se ve agravada por la polarización y el sesgo ideológico de algunas plataformas; por ejemplo, X muestra una tendencia política mayoritariamente conservadora en Estados Unidos. Además, estas aplicaciones emplean algoritmos de recomendación, que seleccionan contenidos en función de sitios visitados, "likes", publicaciones compartidas y datos recopilados a través de cookies, mostrando a los usuarios información ajustada a su perfil, lo que puede generar sesgos y descontextualización del contenido, facilitando la manipulación o el engaño del lector. Se generan una especie de **cámaras de eco** en las redes sociales donde los usuarios únicamente reciben aquellas publicaciones que refuerzan sus creencias, introduciéndolos en su propia burbuja de información.

La sobreinformación y los algoritmos no solo aumentan la cantidad de contenido, sino que además lo filtran de manera que refuerzan sesgos, creando un entorno donde la verdad se mezcla con la falsedad y se hace más difícil de distinguir.

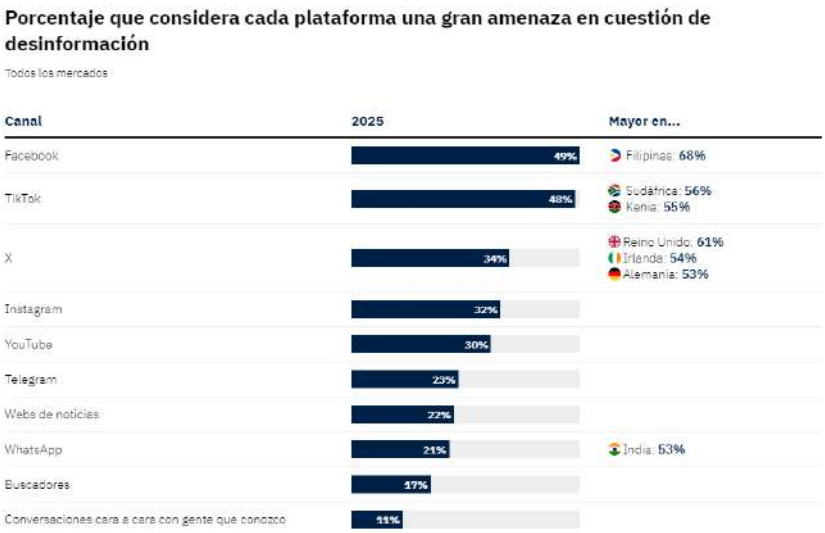


Figura 4: Percepción de difusión de desinformación en cada una de las plataformas

Fuente: Newman (2025)

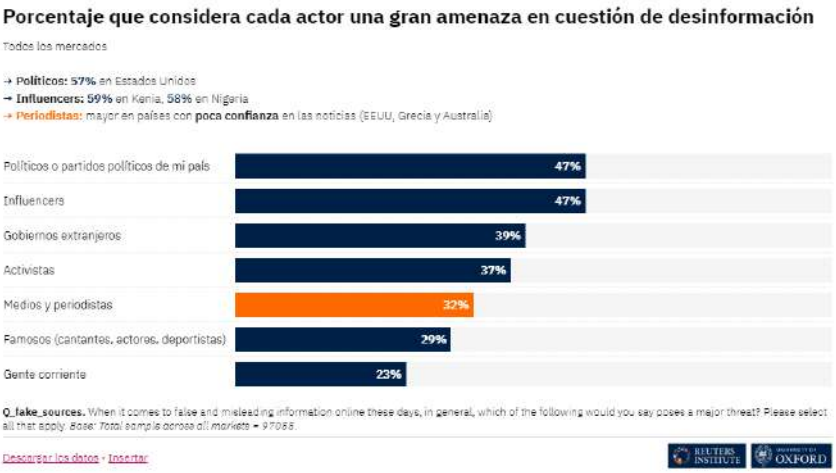


Figura 5: Percepción de difusión de desinformación por cada tipo de escritor

Fuente: Newman (2025)

La **sobrecarga informativa** también contribuye a la dificultad de identificar contenidos verídicos. Los usuarios están expuestos de manera constante a información, opiniones y contenidos audiovisuales que a menudo se contradicen entre sí. Esta exposición, unida a la rapidez e inmediatez del consumo digital, genera lo que se ha denominado “flujo de desinformación” y aumenta los riesgos sociales y empresariales.

Ademas, **la capacidad de concentración y atención** se ha visto reducida, especialmente entre los jóvenes debido a plataformas como TikTok, donde los videos duran apenas 15

segundos, causando que los usuarios rara vez verifiquen la información antes de pasar al siguiente contenido.

Aunque **la desinformación no es un fenómeno nuevo, la era digital ha multiplicado su alcance, velocidad y repercusión**. Internet y las redes sociales permiten que los contenidos falsos o manipulados se difundan más rápido que la información verificada, alterando la forma en que se construye y comparte el conocimiento. Además, los roles tradicionales de emisor y receptor se han difuminado, dando lugar al **prosumidor**, un usuario que no solo consume información, sino que también la produce y distribuye. El lector ya no es un lector pasivo, sino que ademas, genera contenido, amplificando asi la difusión de contenido falso.

La facilidad, bajo coste y rapidez con que hoy se puede difundir información convierte a la desinformación en un desafío global de gran magnitud. Por ello, se hace imprescindible promover estrategias de alfabetización mediática y pensamiento crítico que capaciten a los usuarios para gestionar, filtrar y evaluar la información de manera responsable, fortaleciendo la calidad del ecosistema comunicativo contemporáneo (Oficina C, 2023; Aguaded y Romero-Rodríguez, 2015).

2.2 La desinformación y su topología

La desinformación se define como toda aquella información **que es falsa, parcialmente falsa o descontextualizada, difundida de forma intencionada con el propósito de engañar o manipular a la audiencia** (González Sillot y Martínez Cámara, 2022).

De acuerdo con Baines y Elliott (2020), **la información puede entenderse como la transmisión de conocimiento, sea este veraz o no**. Cuando dicha transmisión contiene elementos falsos o distorsionados, **puede clasificarse en tres categorías**, dependiendo de **la intención del emisor, la veracidad del contenido y la relación entre el mensaje y la realidad.**

Baines y Elliot (2020) entienden que esta la **Misinformation (información errónea),** que es la difusión de información falsa sin intención de engañar. El emisor cree que el contenido es verdadero, pero lo comparte sin verificar su veracidad o sin disponer de información adecuada. Ejemplo: afirmar que el microondas destruye los nutrientes de la comida.

Como segunda categoría podemos encontrar la **Disinformation (desinformación) que es** información deliberadamente falsa, difundida con intención de manipular o influir en la opinión pública. Ejemplo: negar el cambio climático con fines políticos o económicos.

Finalmente, podemos encontrar también la **Malinformation (malinformación),** información veraz que se presenta fuera de contexto o de manera sesgada, provocando una interpretación errónea. Ejemplo: publicar una imagen real de un supermercado vacío en Cuba y afirmar que así están los supermercados en España por miedo a un nuevo apagón.

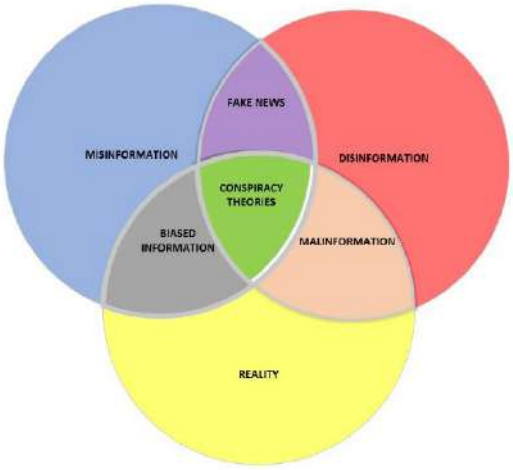


Figura 6: Misinformación, malinformación y desinformación

Fuente: Tătaru, Domenteanu, Delcea, Florescu, Orzan & Cotfas (2024)

Esto evidencia, una vez más, los riesgos asociados a la información contemporánea. No solo existe la desinformación generada de manera intencionada, sino que la difusión de información que consideramos veraz, por ejemplo, al compartir publicaciones en redes sociales sin haberlas verificado por descuido o falta de atención, también contribuye a la propagación de noticias falsas o incluso de información verdadera pero desactualizada.

La desinformación presenta una naturaleza compleja, ya que se puede articular a través de narrativas coherentes y persistentes, mezclando hechos reales con elementos falsos o manipulados, para promover determinados intereses o debilitar la confianza del público en las fuentes de información. Con frecuencia, se adapta al contexto local, cultural o político, lo que dificulta su identificación y aumenta su poder de persuasión. Además, suele difundirse por múltiples canales y formatos, apelando a las emociones del receptor para potenciar su viralización y alcance (Oficina C, 2023).

Es importante destacar que, aunque dicho TFG se vaya a centrar en las Fake News, como indican González Sillot y Martínez Cámara (2022) las noticias falsas no constituyen la única forma de desinformación. También pueden manifestarse mediante rumores, titulares sensacionalistas (clickbait), opiniones manipuladas o contenidos parcialmente verídicos, contribuyendo igualmente a la distorsión del debate público.

2.3 Fake News

2.3.1 Concepto y Características

Según Mao, Hu y Zhang (2025), las **fake news** son un tipo específico de **desinformación** caracterizado por su **creación intencionada, difusión multiformato** (a través de texto, imágenes, videos y principalmente mediante redes sociales, blogs, foros, entre otros) y **recepción controvertida**: algunos usuarios las aceptan, otros las rechazan y muchos las comparten sin verificar su veracidad.

Como se ha mencionado, estas noticias se crean deliberadamente con conocimiento de su falsedad y con el objetivo de engañar a los lectores, manipular la opinión pública, promover intereses determinados o generar beneficios económicos o mediáticos atrayendo la atención del público. Pueden difundirse inicialmente como desinformación (son creadas intencionadamente), pero posteriormente compartirse como misinformación, ya que muchos usuarios desconocen que el contenido es falso.

2.3.2 La eficacia de las Fake news

Según diversos estudios, como el de Wardle y Derakhshan (2017), la eficacia de las fake news depende también de factores cognitivos y socioafectivos.

Entre los **factores cognitivos**, destacan varios fenómenos psicológicos que facilitan la aceptación y difusión de información falsa. El sesgo de confirmación, por ejemplo, hace que las personas tiendan a seleccionar información que confirme sus creencias previas, mientras que el efecto de verdad ilusoria incrementa la probabilidad de aceptar como verdadero un contenido simplemente por su repetición. Asimismo, los individuos pueden confiar excesivamente en fuentes percibidas como confiables, confiar más en la intuición que en el razonamiento crítico o compartir información en redes sociales sin verificarla, ya sea por hábitos, inmediatez o deseo de interacción social.

Por otra parte, los **factores socioafectivos** también juegan un papel crucial en la propagación de las fake news. Estos contenidos suelen apelar a emociones intensas, sensibilidades, creencias o vulnerabilidades del receptor, lo que aumenta su impacto y favorece su viralización. La combinación de elementos emocionales con información distorsionada potencia la probabilidad de que los usuarios la acepten y la difundan, contribuyendo a que las noticias falsas tengan un efecto más profundo y duradero en la sociedad (Oficina C, 2023).

2.3.3 El impacto de las fake news en la sociedad

Las noticias tienen un impacto mayor del que a veces se percibe. La sociedad comparte y difunde información sin ser plenamente consciente de los efectos que dicha información puede generar. Además, una vez que la información falsa ha sido aceptada y se mantiene en el tiempo, **su corrección se vuelve extremadamente difícil**; en algunos casos, intentar rectificarla puede incluso reforzar la creencia en ella (Oficina C, 2023).

Durante la pandemia de COVID-19, por ejemplo, circularon noticias sobre compras masivas de papel higiénico o teorías conspirativas sobre vacunas, como la existencia de microchips para controlar a la población. Estos contenidos contribuyeron a movimientos antivacunas y afectaron la aceptación de la vacunación en distintos grupos. Además, en períodos electorales, las noticias falsas han polarizado la sociedad, afectando la opinión sobre candidatos y resultados. Estos ejemplos muestran cómo la desinformación puede generar caos social, ansiedad, miedo o modificar hábitos diarios de la población.

Más allá de los **efectos sociales**, las noticias falsas también tienen un impacto considerable en el **ámbito empresarial**. la propagación de noticias falsas puede dañar la reputación de una empresa e incluso afectar su valor de mercado. Por ejemplo, entre 2016 y 2017, Apple sufrió la difusión de noticias sobre iPhones que se incendiaban, solicitudes de backdoors por parte del FBI y amenazas de hackers que podrían eliminar millones de cuentas. Como consecuencia, su valoración bursátil disminuyó aproximadamente 0,2 desviaciones estándar (Lin, Zhang, Li, Lin y Liu, 2024).

Cabe destacar que no solo las noticias que afectan directamente a la empresa generan impacto; aquellas relacionadas con productos o servicios también pueden afectar ventas y percepción del mercado. Por ejemplo, la difusión de información falsa sobre que el

chorizo es cancerígeno podría perjudicar negativamente a la industria del embutido, incluso si no hay evidencia científica que lo respalde.

Otro de los impactos relevantes de las noticias falsas se refleja **a nivel macroeconómico**, donde las fake news pueden influir en la imagen y percepción de estabilidad de un país, afectando tanto a la inversión nacional como extranjera. Aunque no modifican directamente las políticas económicas, la incertidumbre y desconfianza que generan pueden impactar la confianza de los mercados, dificultar la implementación de políticas monetarias y fiscales y generar volatilidad en variables como el tipo de cambio. De este modo, la difusión de información falsa puede tener consecuencias económicas indirectas pero significativas, repercutiendo en la estabilidad financiera y en la toma de decisiones de inversores y empresas.

2.3 Tecnologías para la detección de notificas falsas

Dado el impacto significativo de las noticias falsas en la sociedad, resulta fundamental desarrollar **métodos capaces de determinar la veracidad de la información que consumimos**. **A lo largo del tiempo, se han propuesto diferentes enfoques para la detección de noticias falsas** según señalan Jingyuan, Zegiu, Tianyi y Peiyang (2025).

2.3.1 Evolución de los métodos de detección

Como señalan Jingyuan, Zegiu, Tianyi y Peiyang (2025), los métodos para la detección de noticias falsas han experimentado una evolución significativa a lo largo del tiempo. En sus palabras, “The speed and scale of propagation of fake news demand robust automated detection solutions to protect information integrity, democracy, and public trust” [La velocidad y escala de propagación de las noticias falsas exigen soluciones automatizadas robustas para proteger la integridad de la información, la democracia y la confianza pública]. Por esta razón, **se ha pasado de enfoques basados en la extracción manual de características a modelos más complejos**, **como los actuales LLM**, que permiten un análisis más sofisticado y automatizado del contenido informativo.

2.3.2 Enfoques basados en características manuales

En sus inicios, **los primeros modelos se basaban en características diseñadas manualmente**, tales como palabras sospechosas, uso de mayúsculas, puntuación, frecuencia de ciertas URLs, entre otras. Estas características se introducían en modelos de aprendizaje supervisado, como regresión logística, SVM o árboles de decisión, con el objetivo de clasificar las noticias como verdaderas o falsas. Sin embargo, este enfoque presentaba limitaciones importantes: los modelos eran altamente dependientes del conjunto de datos utilizado para su entrenamiento, ya que las características se extraían de manera manual. Como consecuencia, ante cambios en las estrategias de desinformación o variaciones en el estilo de las noticias, su rendimiento se degradaba notablemente.

2.3.3 Aprendizaje profundo

Con la llegada del aprendizaje profundo (Deep Learning), se introdujeron modelos más avanzados capaces de comprender el contexto de las palabras dentro del texto. Entre ellos destacan las RNNs (Redes Neuronales Recurrentes) y las LSTMs (Long Short-Term Memory), que procesan el texto de manera secuencial, palabra por palabra, recordando la información previa y permitiendo así un análisis contextual más completo. Además, **se incorporaron vectores densos de palabras (embeddings)** obtenidos de modelos preentrenados como Word2Vec o GloVe, que representan cada palabra mediante un vector numérico capaz de capturar tanto su significado como su relación con otras palabras del texto. A pesar de estas mejoras, estos modelos sólo capturaban relaciones de contexto cercanas en la secuencia de palabras, limitando la comprensión semántica a corto alcance.

2.3.4 Arquitectura transformer

Posteriormente, surgieron **los modelos basados en la arquitectura de los Transformers**, como BERT y RoBERTa, que adoptaron el mecanismo de atención introducido por Ashish Vaswani et al. (2017). Estos modelos pueden analizar dependencias contextuales dentro del texto e identificar conexiones complejas, lo que ha supuesto un avance significativo en tareas como la detección de noticias falsas, donde resulta esencial captar matices, inferencias y contradicciones sutiles. Su enfoque bidireccional les permite

comprender el significado completo de una frase, lo que mejora la precisión en tareas de clasificación y análisis semántico.

No obstante, los modelos como BERT presentan limitaciones importantes. Su arquitectura original fue diseñada exclusivamente para texto, lo que dificulta el manejo de contenido multimodal (como imágenes, vídeos o gráficos) cada vez más presente en los entornos digitales. Además, su ventana de contexto está limitada (por ejemplo, 512 tokens), lo que restringe el procesamiento de secuencias largas y puede dificultar el análisis de noticias extensas o documentos con múltiples párrafos. Asimismo, requieren ajustes específicos (fine-tuning) para cada tarea, lo que reduce su flexibilidad y complica su aplicación directa en entornos dinámicos como las redes sociales, donde el lenguaje evoluciona rápidamente y la información se propaga de forma caótica.

2.3.5 Transición hacia los modelos de lenguaje de gran tamaño (LLM)

Estas limitaciones llevaron al desarrollo de los **modelos de lenguaje grande (LLMs),** que extienden la arquitectura de los Transformers mediante entrenamiento a gran escala con enormes volúmenes de datos. Gracias a este enfoque, los LLMs presentan una mayor capacidad de generalización y adaptabilidad, lo que les permite abordar múltiples tareas sin necesidad de reentrenamiento específico, aumentando su robustez en la detección de desinformación en entornos complejos y dinámicos. A diferencia de modelos como BERT, donde el fine-tuning es obligatorio para adaptar el modelo a tareas concretas, los LLMs pueden funcionar directamente mediante prompting, aunque el fine-tuning sigue siendo opcional para mejorar su rendimiento o alinear sus respuestas con criterios específicos. Según Naveed et al. (2023), “These developments have brought about a revolutionary transformation by enabling the creation of LLMs that can approximate human-level performance on various tasks” [estos desarrollos han traído una transformación revolucionaria, permitiendo crear LLM que pueden aproximarse al rendimiento humano en diversas tareas]. Además, las variantes multimodales de LLMs, capaces de procesar y generar contenido que combina texto con otros formatos como imágenes o audio, amplían su aplicabilidad en escenarios donde la desinformación se presenta en múltiples formas.

Aunque BERT comparte elementos arquitectónicos con los LLM modernos y fue entrenado con grandes corpus de texto, su arquitectura es de tipo encoder-only, lo que le

permite comprender texto, pero no generarlo. Por ello, no suele considerarse un LLM completo en los estudios recientes, ya que los criterios típicos incluyen decenas de miles de millones de parámetros, entrenamiento a gran escala y capacidad generativa o multitarea. Por tanto, conviene tratarlo como un "modelo preentrenado grande" más que como un LLM generativo completo (Naveed et al., 2023).

2.4 Procesamiento de lenguaje natural (Natural processing)

Antes de comprender qué es un modelo de lenguaje de gran tamaño (LLM), es necesario entender primero el concepto de Procesamiento del Lenguaje Natural (PLN o NLP). Según Kuiler (2021), esta disciplina, enmarcada dentro del ámbito de la inteligencia artificial (IA), **combina técnicas de machine learning y deep learning para dotar a los sistemas informáticos de la capacidad de interpretar, analizar y generar lenguaje humano**.

El NLP se articula en dos grandes áreas funcionales, que permiten que las maquinas interactúen con el lenguaje de forma más natural y contextualizada. Por un lado, esta el **Natural Language Undersatanding (NLU),** que esta enfocada en la comprensión del lenfuaje y, por otro lado, esta el **Natural Language Generation (NLG),** que esta orientada a la generación automática de texto.

Figura 7: Natural language processing

Fuente: Bird, Klein & Loper (2009)

Una de las principales fortalezas del NLP es su capacidad para **trabajar con grandes volúmenes de texto no estructurado, extrayendo información relevante y generando contenido de manera autónoma**. Para lograrlo, el NLP se apoya en fundamentos

lingüísticos que incluyen la morfología, la sintaxis, la semántica y la pragmática. Este último aspecto (pragmática) es fundamental para interpretar correctamente el sentido de las palabras según la situación comunicativa.

Durante el procesamiento textual, el NLP realiza una serie de tareas básicas: segmentación en oraciones, división en unidades mínimas (tokens), lematización, eliminación de términos irrelevantes (stopwords), análisis de relaciones sintácticas y reconocimiento de entidades (nombres propios, lugares, fechas), asignar a cada palabra su categoría gramatical (como sustantivo, verbo o adjetivo), o resolver referencias cruzadas (como identificar a quien se refiere la palabra "ella"). Estas operaciones permiten llevar a cabo análisis más avanzados, como generación de resúmenes automáticos, detección de emociones o sentimientos, identificación de distintos puntos de vista y extracción de los temas principales tratados en un conjunto de documentos.

Estas capacidades constituyen la base sobre la que se desarrollan los LLM, que amplían y automatizan estas funciones a gran escala para producir y comprender texto de manera eficiente. (Kuiler, 2021)

Dentro de los modelos explicados anteriormente, tanto los LLM (incluyendo BERT) como los embeddings son técnicas / modelos dentro del campo del NLP.

2.5 Modelos de lenguaje de gran tamaño (LLM)

2.5.1 Introducción a los LLM

Como indican Naveed et al. (2021), los LLM son sistemas de inteligencia artificial de última generación que se basan en arquitecturas de redes neuronales (fundamentalmente Transformers) para procesar, comprender y generar lenguaje natural de una manera similar a como lo hace un ser humano, permitiendo una "comunicación coherente". Tal como señalan Naveed et al., los LLM han demostrado "Large Language Models (LLMs) have recently demonstrated remarkable capabilities in natural language processing tasks and beyond" [Capacidades destacadas en tareas de procesamiento del lenguaje natural y más allá] (como razonamiento, generación de código, traducción) gracias al gran tamaño del modelo, los datos de entrenamiento y la arquitectura eficiente.

En el marco de este trabajo, su principal ventaja es que pueden entender el contexto completo de una noticia (por ejemplo, detectar contradicciones internas, matices

implícitos, estilo de redacción variable), lo que les permite analizar su contenido de manera más precisa. Gracias a esta capacidad, los LLM pueden detectar noticias falsas incluso cuando cambia el estilo de redacción o el tema de las noticias, requiriendo muy poca supervisión y sin necesidad de entrenar un modelo desde cero para cada nuevo caso (Jingyuan, Zegiu, Tianyi y Peiyang, 2025).

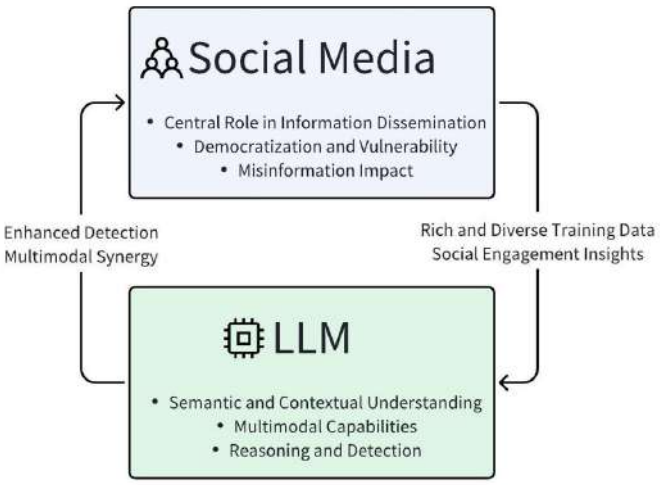


Figura 8: cómo funciona un LLM con redes sociales

Fuente: Jingyuan, Zegiu, Tianyi & Peiyang, 2025

Como se mencionó previamente, la transformación revolucionaria de los LLM se sustenta principalmente en tres factores: la arquitectura Transformer, que constituye la base fundamental de estos modelos; el incremento de las capacidades computacionales, dado que un modelo se considera LLM cuando cuenta con al menos 10 mil millones de parámetros; y el entrenamiento a gran escala, que utiliza datasets de terabytes de texto para que el modelo aprenda patrones lingüísticos complejos y pueda generalizar a múltiples tareas (Naveed et al. , 2021).

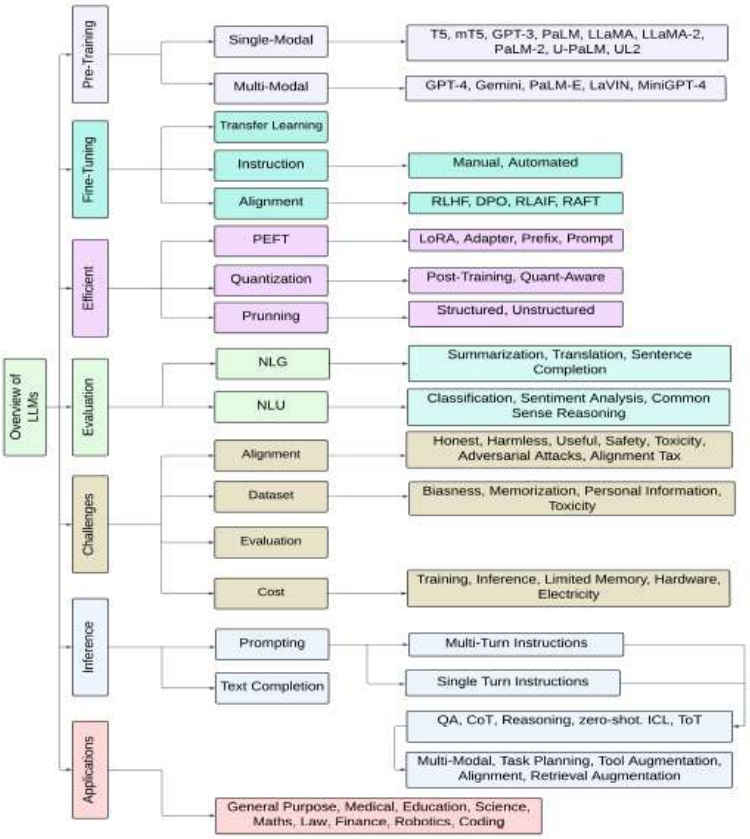


Figura 9: Overview of LLM's

Fuente: Naveed et al (2021)

2.5.2 Componentes principales de un LLM

Para funcionar correctamente, Naveed et al. (2021), indican que los LLM cuentan con los siguientes componentes;

Los **tokens**, que dividen el texto en unidades (tokens) descomponibles que el modelo puede procesar (por ejemplo, WordPiece, Byte Pair Encoding), los **mecanismo de atención** que "Assigns weights to input tokens based on importance so that the model gives more emphasis to relevant tokens" [asigna pesos a los tokens de entrada basándose en su importancia, de modo que el modelo enfatiza los tokens más relevantes], la **codificación de posición**, que informa al modelo de la posición de cada token, permitiendo procesar el texto en paralelo sin perder la información secuencial, las **funciones de activación,** que determinan cómo se transforma la señal en cada neurona para introducir no-linealidad (por ejemplo, ReLU, GeLU) y la **normalización de capas / Layer normalization** que ajusta y normaliza los valores de activación de las neuronas

(media cero, desviación estándar uno) para que el aprendizaje sea más estable y no se vea afectado por valores extremos.

2.5.3 Arquitectura basada en Transformers

El desarrollo de los LLM se ha dado gracias a la arquitectura de los Transformers (Vaswani et al., 2017). Esta arquitectura permite procesar los datos de forma paralela y aprender las relaciones entre palabras, sin importar la distancia entre ellas en el texto. Esto posibilita que los LLM entiendan el contexto completo, analizando todo el texto a la vez, y no únicamente palabra por palabra.

Como indican Naveed et al. (2021), existen distintas variantes de Transformers según la aplicación deseada.

Por un lado, se encuentran los **Transformers encoder-decoder**, en los que el encoder procesa toda la información de entrada y comprende el contexto completo antes de pasarlo al decoder, que genera la salida alineada con la entrada. Ejemplos de esta arquitectura son **T5** y **UL2**.

También existen arquitecturas que consisten únicamente en **encoders** o únicamente en **decoders**. Los encoders solos pueden procesar la entrada y generar representaciones internas, pero no producen directamente una salida interpretable en forma de secuencia. Por ello, rara vez se consideran una arquitectura completa para tareas de generación, y normalmente requieren un decoder que transforme estas representaciones en resultados concretos.

En cuanto a los decoders, se distinguen dos tipos de arquitectura: el **Causal Decoder** o decodificador causal, que no tiene encoder explícito y genera la salida palabra por palabra, considerando únicamente los tokens previamente generados, como ocurre en **GPT-3** o **LLaMA**; y el **Prefix Decoder**, donde la atención del decoder es bidireccional y puede atender tanto a información pasada como futura dentro de la secuencia.

Finalmente, también existen los **Mixture of Experts (MoE)**, compuestos por varios expertos independientes, como capas feed-forward especializadas, y un **router** que decide qué experto se activa para cada token, optimizando así el uso de recursos en modelos de gran escala.

2.5.4 Funcionamiento de los LLM

Naveed et al. (2021), explican que el funcionamiento de los LLM se articula en tres etapas principales, desde su entrenamiento inicial hasta su uso final por parte del usuario.

La primera etapa es el **Pre-entrenamiento (Pre-training)**. En esta fase, el modelo se entrena de forma auto-supervisada, sin necesidad de etiquetas humanas, mediante la exposición a un gran corpus de texto muy diverso. Este entrenamiento le permite aprender el significado de las palabras, la gramática, el estilo y el funcionamiento general del lenguaje. Por ejemplo, puede entrenarse para predecir el siguiente token dado un conjunto de entrada (modelado de lenguaje completo) o para determinar un token intermedio a partir de tokens anteriores y posteriores (modelado de lenguaje enmascarado / masked language modelling).

Además, como destacan Naveed et al., las *scaling laws* muestran que el desempeño de los LLM sigue una relación predecible con el número de parámetros, la cantidad de datos de entrenamiento y la capacidad computacional disponible. Estas leyes permiten estimar qué tamaño de modelo y qué volumen de datos son necesarios para alcanzar un nivel de rendimiento específico. Asimismo, indican que, por ejemplo, al duplicar los parámetros del modelo, también es necesario duplicar la cantidad de tokens utilizados en su entrenamiento para lograr mejoras en el rendimiento.

En ocasiones, al entrenar LLM a gran escala, el modelo no puede caber en una sola máquina, por lo que se requieren infraestructuras distribuidas que permiten dividir tanto los datos como los parámetros entre múltiples dispositivos, lo que posibilita escalar a miles de millones de parámetros.

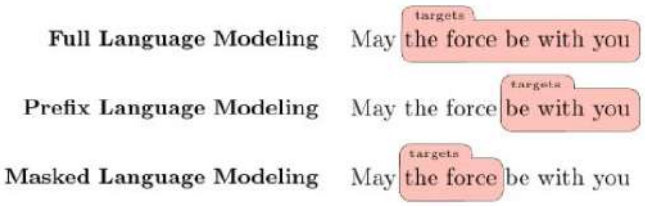


Figura 10: Ejemplo de entrenamiento de un modelo de lenguaje

Fuente: Naveed et al (2021)

Tras el pre-entrenamiento se realiza el **fine-tuning** con datos etiquetados para adaptar el modelo a tareas específicas, mejorar su precisión y asegurar que sigan correctamente la intención del usuario y no caiga en errores de generalización. Existen distintos tipos de fine-tuning: por ejemplo, el *instruction tuning* enseña al modelo a responder

correctamente a instrucciones concretas, y la *alineación (alignment)* busca que el modelo siga las preferencias humanas y evite respuestas incorrectas o dañinas, normalmente mediante técnicas como el RLHF (Reinforcement Learning with Human Feedback) en la que personas califican la calidad de las respuestas del modelo. Gracias a este proceso, incluso ante preguntas sensibles, como las relacionadas con autolesiones o suicidio, el LLM puede ofrecer respuestas seguras y responsables, sugiriendo, por ejemplo, que la persona busque ayuda profesional o apoyo de familiares o de seres de confianza.

Finalmente, el LLM está listo para su uso mediante **consultas (prompts).** El prompting puede consistir en una pregunta sin ejemplos previos donde el modelo debe responder algo que no ha visto antes (zero-shot prompting), incluir ejemplos (few-shot prompting) o razonamientos explícitos (reasoning in LLM like Chain of thoughts) que ayudan al modelo a generar respuestas más precisas. Estas últimas modalidades permiten ajustar el rendimiento del LLM sin necesidad de reentrenamiento, lo que resulta especialmente útil cuando se trabaja con modelos ligeros o se dispone de recursos computacionales limitados. Además, al definir las instrucciones para el LLM, podemos hacerlo mediante single-turn instructions, donde toda la información se proporciona de una sola vez en un único prompt (ya sea en modo zero-shot, few-shot o con razonamiento) o también podemos optar por multi-turn instructions, en las que el modelo recibe retroalimentación y múltiples entradas a lo largo de varias interacciones, lo que le permite comprender mejor el contexto y generar una salida más precisa.

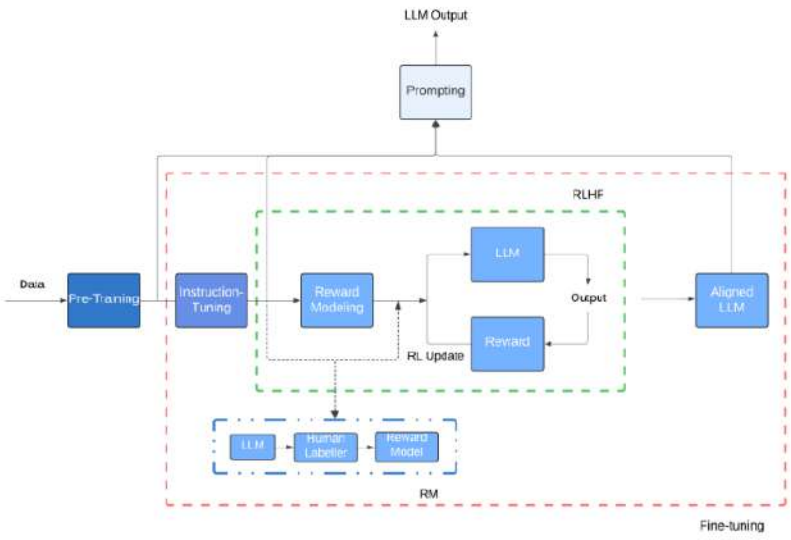


Figura 11: Funcionamiento de un LLM

Fuente: Naveed et al (2021)

3. Investigación

3.1 Introducción del trabajo de investigación

Como se ha mencionado**, anteriormente, se empleaban modelos clásicos de aprendizaje automático**, como la regresión logística, las máquinas de soporte vectorial (SVM) o los árboles de decisión, para entrenar modelos capaces de determinar si una noticia es verdadera o falsa. Sin embargo, con los avances tecnológicos y la evolución del Procesamiento del Lenguaje Natural (NLP), especialmente con la aparición de arquitecturas como los Transformers, se ha producido **un progreso significativo hacia técnicas más avanzadas** que permiten realizar estas tareas con mayor precisión, **como los modelos de lenguaje de gran tamaño (LLM)**.

En este Trabajo de Fin de Grado s**e ha seleccionado una base de datos y se han comparado distintos modelos de clasificación con el objetivo de analizar si, efectivamente, estas técnicas más recientes (los LLM) proporcionan una mejora en la detección de noticias falsas.**

En dicha comparativa, el análisis se ha centrado en modelos que utilizan el procesamiento de lenguaje natural (NLP), es decir, modelos basados en **embeddings** y en modelos de lenguaje de gran escala (**LLM**), omitiendo deliberadamente técnicas iniciales más simplistas, como los árboles de decisión basados únicamente en metadatos (fuente, longitud o ratio de puntuación). Esta decisión se fundamenta en que dichas técnicas no realizan un análisis intrínseco de la noticia, sino una evaluación de su **maquetación y entorno**.

Además, estos modelos simplificados presentan una alta **fragilidad ante ataques adversarios**: un generador de desinformación podría evadir fácilmente la detección simplemente corrigiendo el formato o la puntuación de su texto. Por el contrario, los modelos de **embeddings** permiten representar el significado léxico en un espacio vectorial de similitudes, mientras que los **LLM** logran interpretar el texto de forma contextual, construyendo representaciones internas que permiten un **razonamiento profundo** sobre la coherencia y veracidad del mensaje, lo que los hace significativamente más robustos y fiables.

El objetivo es comprobar si los modelos funcionan para la detección de noticias falsas y si los avances en el ámbito del **Procesamiento del Lenguaje Natural (NLP)**, que han dado lugar a los **Modelos de Lenguaje de Gran Tamaño (LLM)**, conllevan mejoras

sustanciales más allá de un aumento en la sofisticación o complejidad de los modelos. En el contexto de la detección de noticias falsas, la principal diferencia entre los enfoques basados en embeddings y los LLM radica en su capacidad de comprensión del lenguaje.

Los modelos basados en embeddings representan los textos como **vectores numéricos en un espacio semántico**, lo que les permite identificar similitudes de significado incluso cuando no hay coincidencia léxica directa. Sin embargo, presentan limitaciones a la hora de captar el contexto profundo, la ironía, el doble sentido o el significado implícito del lenguaje. Por el contrario, los LLM muestran una **mayor capacidad para interpretar el contexto, inferir relaciones semánticas complejas y aproximarse a formas de razonamiento lingüístico**, lo que les permite abordar fenómenos como la ironía, el sarcasmo o el humor de manera más eficaz.

Para ello, se ha utilizado una base de datos denominada **The Spanish Fake News Corpus**, disponible en GitHub. Montserrat, Posadas, Bel y Porto (2021) señalan que este corpus está compuesto por noticias recopiladas entre noviembre de 2020 y marzo de 2021, todas ellas en español. Estas noticias proceden de diferentes países, como España, Argentina, Bolivia, Chile, Colombia, Costa Rica, Ecuador, Estados Unidos, Francia, Perú, Uruguay, Inglaterra, Venezuela y México, lo que permite incluir diferentes variantes del español y, por tanto, facilita que los modelos desarrollados sean capaces de adaptarse a distintos estilos de escritura.

Además, el corpus no solo incluye noticias procedentes de medios de comunicación tradicionales, sino también de redes sociales, ya que en la actualidad estas plataformas constituyen una de las principales vías de difusión de noticias falsas. En concreto, el 15,73% de las noticias del corpus provienen de redes sociales. El conjunto de datos está formado por 1543 noticias, tanto verdaderas como falsas, que abarcan diferentes temáticas como ciencia, deportes, política, sociedad, COVID-19, medio ambiente e internacional.

La base de datos se ha trabajado en un archivo Excel que contiene los siguientes campos:

- **ID**: número identificativo de cada noticia.
- **Category**: indica si la noticia es verdadera o falsa.
- **Topic**: temática de la noticia.
- **Source**: medio o plataforma en la que fue publicada la noticia.

- **Headline**: título de la noticia.
- **Text**: contenido completo de la noticia.
- **Link**: enlace a la fuente original.

3.2 Preparación de datos

El primer paso realizado con la base de datos ha sido su **preparación y limpieza**. El conjunto de datos original se encontraba dividido en tres archivos Excel diferentes: **train**, **test** y **development**. El conjunto **train** representa aproximadamente el 44 % del total de datos, el **test** el 37 % y el **development** el 19 %.

Al analizar los distintos conjuntos de datos se detectaron algunas inconsistencias que hicieron necesario realizar un **proceso de limpieza previo** para poder trabajar adecuadamente con ellos. Por ejemplo, en el conjunto test se observa que existen 572 noticias registradas; sin embargo, únicamente 565 de ellas disponen de información en el campo source, 500 cuentan con headline y 569 incluyen link. La presencia de estos **valores faltantes** puede dificultar el tratamiento de los datos y provocar errores durante el entrenamiento de los modelos. Por este motivo, se decidió eliminar aquellas filas que contenían campos vacíos en variables relevantes.

Durante este análisis también se observó que **la mayoría de las noticias que presentan valores faltantes corresponden a noticias clasificadas como falsas (en concreto, 73/75).** Este hecho podría interpretarse como un posible patrón, ya que **las noticias incompletas o con información limitada pueden estar asociadas a fuentes menos fiables.** No obstante, para evitar introducir sesgos adicionales en el modelo y garantizar un tratamiento homogéneo de los datos, se optó por eliminar dichas filas en lugar de utilizar la ausencia de información como una característica de clasificación. De esta manera, se asegura que los modelos se centren en aprender a partir del **contenido textual y las características reales de la noticia**, evitando que la clasificación dependa de la presencia o ausencia de datos en los campos del dataset. Además, **se asume que los lectores suelen percibir una noticia con elementos faltantes como potencialmente poco profesional o incompleta**, lo que refuerza la idea de que la información ausente no debería ser utilizada como criterio automático de clasificación, sino que su eliminación permite un análisis más riguroso y generalizable.

Dado que el conjunto **development** no era estrictamente necesario para el proceso experimental planteado en este trabajo, se optó por unificar los tres archivos originales en un único conjunto de datos y realizar posteriormente una nueva partición, generando un **80 % de datos para entrenamiento (train)** y un **20 % para prueba (test)** para asi disponer de conjuntos de entrenamiento y test más amplios.

Asimismo, durante el proceso de exploración de los datos se detectó que en el conjunto development existían noticias con títulos repetidos y otras con el mismo contenido textual. Por ello, antes de establecer el conjunto de datos final, se llevó a cabo una **verificación para identificar posibles duplicados completos** en las filas del dataset. En caso de detectarse registros idénticos, se procedería a eliminar uno de ellos con el fin de evitar redundancias en el entrenamiento de los modelos. Sin embargo, este no era el caso.

Finalmente, al seleccionar estos dos conjuntos, resulta fundamental garantizar que el dataset de entrenamiento contenga una **proporción equilibrada de noticias verdaderas y falsas**. Esto se debe a que, si el conjunto estuviese desbalanceado, por ejemplo, con un 90 % de noticias verdaderas y un 10 % de noticias falsas, el modelo podría aprender un sesgo hacia la clase mayoritaria, clasificando por defecto todas las noticias como verdaderas y logrando aparentemente un alto porcentaje de acierto (90 %), aunque sin aprender a distinguir correctamente entre ambas clases.

Para evitar este problema, en nuestro caso hemos equilibrado los datos, obteniendo un **53 % de noticias verdaderas y un 47 % de noticias falsas**, lo que reduce significativamente la posibilidad de que el modelo incurra en este tipo de error.

3.3 Análisis exploratorio de los datos

Antes de desarrollar un modelo de detección de noticias falsas, se llevó a cabo un **análisis exploratorio de los datos** (EDA, por sus siglas en inglés), que permite obtener una primera visión sobre el comportamiento de las noticias falsas frente a las verdaderas y **detectar patrones relevantes en los datos**. Este análisis resulta especialmente útil para la construcción y ajuste de modelos de clasificación más tradicionales, los cuales constituyen una de sus principales limitaciones: requieren una selección manual de característica, limpieza cuidadosa de texto, preprocesamiento y un entrenamiento más extenso para alcanzar un rendimiento adecuado.

Durante el análisis exploratorio se estudiaron las frecuencias de aparición de palabras, los términos más comunes, así como los sentimientos y emociones asociados a las noticias y el estilo de escritura mediante el uso de metadatos. A partir de este estudio, se pudieron identificar patrones preliminares que pueden orientar la selección de características y estrategias de preprocesamiento para los modelos de clasificación.

Es importante destacar que todos estos análisis se realizaron **únicamente sobre los datos del conjunto de entrenamiento**, con el fin de evitar **data leakage** y garantizar que los datos de prueba permanezcan inéditos. Esto permite evaluar los modelos de manera objetiva y evitar que los resultados se vean artificialmente inflados debido al conocimiento previo de los datos de test y se garantiza que la evaluación de los modelos sobre el test refleje su rendimiento real y generalizable.

3.3.1. Análisis de frecuencias

Dentro de este análisis se examinó si existían diferencias significativas en función de las **fuentes** de las noticias y de los **tópicos** o temáticas que abordan.

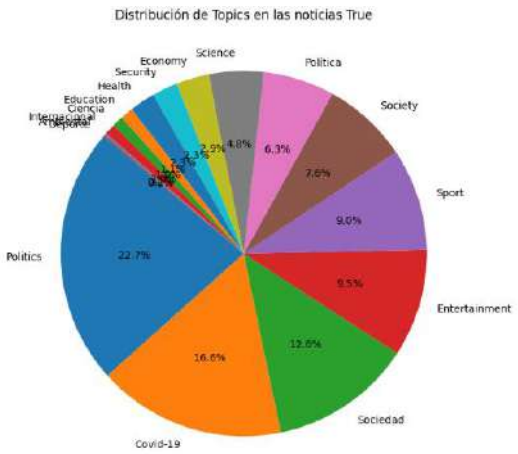


Figura 12: Frecuencia de los temas en las noticias True

Fuente: Elaboración propia

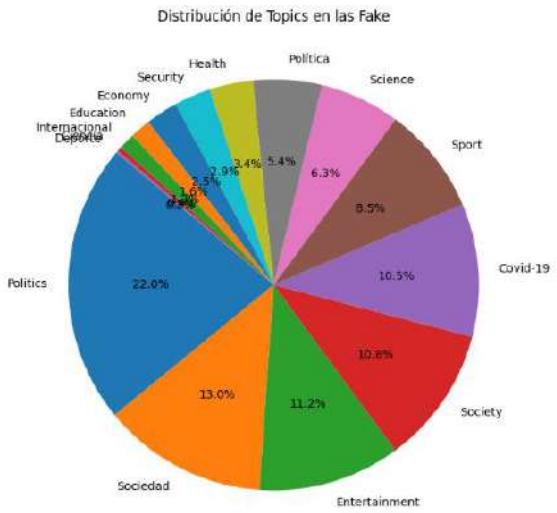


Figura 13: Frecuencia de los temas en las noticias Fake

Fuente: Elaboración propia

Se puede concluir que, **aunque existen pequeñas diferencias, tanto las noticias falsas como las verdaderas abordan principalmente los mismos temas**, siendo los más recurrentes en ambos casos: política, COVID-19, sociedad, entretenimiento y deportes. No obstante, estas diferencias temáticas probablemente reflejan más la popularidad o relevancia de ciertos tópicos durante el periodo de recopilación de los datos que la veracidad de la noticia en sí. Por esta razón, no se consideró oportuno incorporar la categoría temática como variable en los modelos de clasificación, ya que se estima que no aporta un valor predictivo significativo.

En cambio, se observa que la **fuente de la noticia** presenta una diferencia más marcada entre noticias falsas y verdaderas, lo que sugiere que **este atributo podría constituir uno de los factores más relevantes a la hora de determinar la veracidad de una noticia**.

La figura 14 presenta, para cada fuente incluida en el dataset de entrenamiento, el número de noticias verdaderas y falsas publicadas. Las noticias verdaderas se representan en color verde, mientras que las noticias falsas se muestran en color rojo, lo que permite visualizar de manera inmediata la distribución de la veracidad de las noticias según su origen. Al examinar el gráfico con mayor detalle, se puede observar que la gran mayoría de las fuentes tiende a publicar únicamente un tipo de noticia, ya sea verdadera o falsa. Solo un total de 11 fuentes incluyen noticias de ambas categorías, lo que evidencia que la fuente puede constituir un **indicador relevante** a la hora de determinar la veracidad de una

noticia.

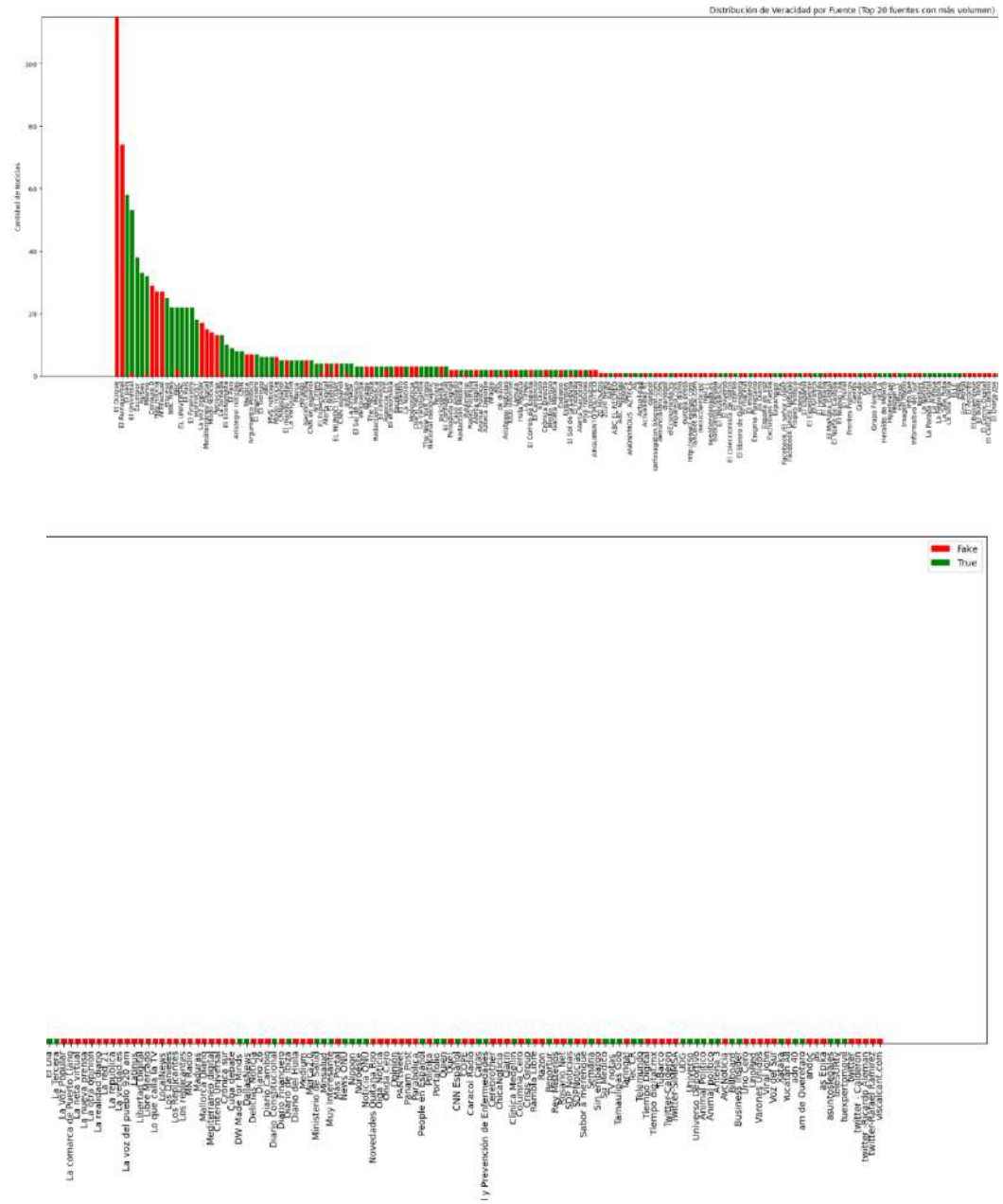


Figura 14: Noticias verdaderas y falsas por cada fuente

Fuente: Elaboración propia

3.3.2 DTM y TF-IDF

En este análisis se estudiaron tanto los **títulos** como los **textos completos** de las noticias con el objetivo de identificar términos que se repitan de manera destacada en alguno de los conjuntos. Para ello, se llevó a cabo un preprocesamiento del texto que incluyó la eliminación de **stopwords**, la conversión de todas las palabras a **minúsculas** y la

supresión de caracteres especiales, conservando únicamente letras y caracteres acentuados del español mediante la expresión regular [^a-zA-ZáéíóúüñÁÉÍÓÚÜÑ].

Una vez realizado este preprocesamiento, se construyó una **matriz término-documento (DTM, Document-Term Matrix)** que permite determinar la frecuencia con la que cada término aparece en los distintos documentos, facilitando la identificación de palabras representativas de noticias falsas o verdaderas.

```
number       2621
méxico        633
si            596
años          584
personas      503
ser           500
país          482
dijo          460
gobierno      435
dos           420
covid         382
parte         354
presidente    350
puede         338
año           335
después       332
según         329
así           327
millones      322
salud         308
dtype: int64
```

Figura 15: Palabras más comunes en las noticias true

Fuente: Elaboración propia

```
number       876
si           514
méxico       396
ser          361
así          302
años         271
país         252
ahora        234
gobierno     230
además       228
personas     224
solo         221
presidente   216
según        207
dijo         205
puede        194
vez          186
parte        186
virus        180
hace         178
dtype: int64
```

Figura 16: Palabras más comunes en las noticias fake

Fuente: Elaboración propia

Al analizar los términos más frecuentes, se observa que las palabras más comunes en las noticias verdaderas son similares a las de las noticias falsas, lo que indica que este análisis no revela **hallazgos relevantes** que puedan diferenciar claramente ambos conjuntos. De manera análoga, el estudio de los **títulos de las noticias** muestra que los términos más repetidos también son muy similares entre noticias verdaderas y falsas, por lo que este enfoque no aporta información discriminativa significativa para la clasificación.

```
...  covid          64
     number         44
     coronavirus    34
     méxico         34
     amlo           32
     trump          17
     tras           15
     vacuna         15
     peña           15
     cómo           14
     vacunas        13
     eu             12
     años           12
     mujeres        12
     gobierno       11
     salinas        11
     meade          11
     nuevo          11
     mundo          10
     primera        10
     dtype: int64
```

Figura 17: Palabras más comunes en los titulares true

Fuente: Elaboración propia

```
...  number         54
     amlo           43
     méxico         34
     covid          19
     gobierno       19
     años           19
     coronavirus    18
     peña           17
     tras           15
     meade          14
     rivera         14
     pri            13
     nieto          13
     nueva          13
     si             13
     millones       12
     trump          12
     ahora          12
     nuevo          11
     américa        11
     dtype: int64
```

Figura 18: Palabras más comunes en los titulares fake

Fuente: Elaboración propia

Se llevó a cabo un análisis adicional aplicando **lemmatización** a las palabras, proceso mediante el cual cada término se reduce a su **lema**, con el objetivo de disminuir la dimensionalidad de la matriz término-documento (DTM) y minimizar el "efecto" del estilo de redacción de cada autor, centrándose únicamente en el significado de las palabras. Durante este proceso, se identificaron y eliminaron términos que aparecían con frecuencia pero no aportaban valor informativo, como el, meadir, peño o ir.

A pesar de aplicar la lematización y de eliminar estos términos poco relevantes, el análisis **no permitió identificar información discriminativa ni diferencias significativas en el vocabulario empleado entre noticias verdaderas y falsas**. Esto sugiere que la simple frecuencia de palabras, incluso tras aplicar técnicas de normalización y reducción de dimensionalidad, no proporciona un criterio útil para diferenciar ambos tipos de noticias.

```
Top 20 Lemas headlines true (Limpieza Radical):
covid          64
méxico         34
amlo           32
coronavirus    28
trump          17
mujer          17
vacuna         14
pedir          14
elección       13
mexicano       11
peña           11
país           11
muerte         11
gobierno       11
nieto          10
político       10
millón         10
mundo           9
recibir         9
carmen          9
dtype: int64
```

Figura 19: Lemas más comunes en los titulares de las noticias true

Fuente: Elaboración propia

```
Top 20 Lemas headlines false (Limpieza Radical):
amlo           43
méxico         34
mexicano       22
mujer          21
covid          19
gobierno       19
coronavirus    14
nieto          14
pri            13
revelar        13
millón         12
pedir          12
morir          12
país           12
trump          12
facebook       11
español        11
recibir        11
permitir       10
nombre         10
dtype: int64
```

Figura 20: Lemas más comunes en los titulares de las noticias fake

Fuente: Elaboración propia

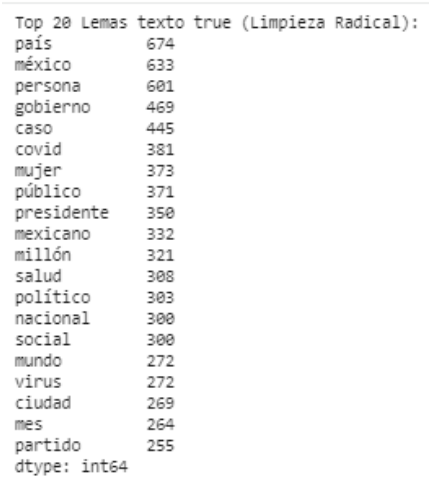


Figura 21: Lemas más comunes en las noticias true

Fuente: Elaboración propia

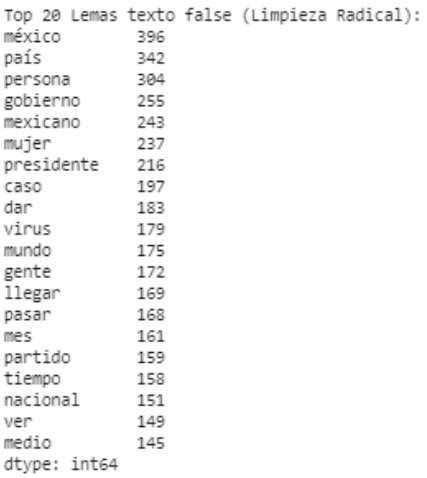


Figura 22: Lemas más comunes en las noticias fake

Fuente: Elaboración propia

Finalmente, se realizó un **análisis mediante TF-IDF** sobre los lemas, con el objetivo de evaluar la importancia relativa de los términos dentro de cada corpus (noticias verdaderas o falsas). Esta métrica permite identificar qué palabras o significados son más representativos en cada conjunto. No obstante, al igual que en los análisis previos, **se llegó a la conclusión de que los términos, significados y palabras utilizados no presentan**

diferencias significativas entre noticias verdaderas y falsas, por lo que este enfoque no proporciona información discriminativa útil para la clasificación.

Top 20 Significados más distintivos en texto true:

	término	importancia
1250	méxico	0.027558
1379	país	0.025354
1403	persona	0.022998
895	gobierno	0.020634
1483	presidente	0.019976
265	caso	0.019715
1156	mexicano	0.019665
1778	social	0.018316
1566	público	0.018043
1257	nacional	0.017759
445	covid	0.017431
299	ciudad	0.016479
1238	mujer	0.016453
1205	millón	0.016441
1371	partido	0.016187
1193	mes	0.016050
1448	político	0.015913
1706	salud	0.015900
1242	mundo	0.015635
1605	red	0.015450

Figura 23: Significados más comunes en las noticias true

Fuente: Elaboración propia

Top 20 Significados más distintivos en texto fake:

	término	importancia
1257	méxico	0.027047
1393	país	0.023849
1424	persona	0.020602
1197	mexicano	0.019649
869	gobierno	0.018700
1504	presidente	0.018573
1242	mujer	0.017443
264	caso	0.016624
489	dar	0.015973
1246	mundo	0.015717
859	gente	0.015284
1007	llegar	0.015251
1387	pasar	0.015249
1856	tiempo	0.014929
1939	ver	0.014928
1383	partido	0.014713
1792	social	0.014431
1456	poder	0.014411
1263	nacional	0.014319
1309	obrador	0.014230

Figura 24: Significados más comunes en las noticias fake

Fuente: Elaboración propia

Top 20 Significados más distintivos en headlineas fake:

	término	importancia
45	amlo	0.020446
1714	méxico	0.017464
1684	mexicano	0.011794
696	covid	0.011066
1704	mujer	0.010428
1546	gobierno	0.010228
674	coronavirus	0.009560
1699	morir	0.008279
1725	nieto	0.007144
1856	revelar	0.007132
1812	pri	0.007079
1768	país	0.006870
1840	recibir	0.006657
1770	pedir	0.006649
1559	guerra	0.006641
1778	permitir	0.006401
1304	español	0.006307
1429	facebook	0.006280
939	descubrir	0.005998
59	angélico	0.005992

Figura 25: Significados más comunes en los titulares fake

Fuente: Elaboración propia

Top 20 Significados más distintivos en headlines true:

	término	importancia
782	covid	0.023813
53	amlo	0.015462
1723	méxico	0.014986
747	coronavirus	0.012938
1925	trump	0.009161
1716	mujer	0.008706
1769	pedir	0.007556
1933	vacuna	0.007294
1719	mundial	0.007024
1715	muerte	0.006565
1700	mexicano	0.006308
1249	elección	0.006056
1767	país	0.006005
70	aprobar	0.005977
1609	gobierno	0.005812
1668	llegar	0.005784
1785	peña	0.005635
1624	hijo	0.005488
1713	morir	0.005438
367	carmen	0.005348

Figura 26: Significados más comunes en los titulares true

Fuente: Elaboración propia

Dado que esta información ya se encuentra implícita en el análisis del contenido textual y no aporta un valor adicional para la detección de noticias falsas, no se incorporará nuevamente al momento de generar la entrada utilizada para la clasificación de las noticias.

Finalmente, se realizó **un análisis de la densidad de stopwords**, obteniendo una proporción **del 42,23% para noticias falsas** y del **41,21% para las verdaderas**. Dada la similitud estructural entre ambos conjuntos y la escasa diferencia observada (1,02%), se optó por no eliminar las stopword*s* en la fase de preprocesamiento, ya que no muestran un poder disciminativo.

Esta decisión se justifica, además, porque los modelos basados en embeddings se benefician de la preservación del contexto completo, incluyendo palabras funcionales que contribuyen a la estructura sintáctica y al significado global del texto, evitando así posibles pérdidas de información relevantes.

3.3.3 Análisis de sentimientos y emociones

En la redacción de noticias, salvo en textos de opinión, generalmente se busca mantener un **tono neutro y objetivo,** mientras **que las noticias falsas buscan apelar a las emociones de los lectores**; sin embargo, dado que los autores son seres humanos, este objetivo no siempre se cumple plenamente. Por ello, se procedió a analizar tanto las **seis emociones básicas** (alegría (*joy*), ira (*anger*), tristeza (*sadness*), sorpresa (*surprise*),

miedo (*fear*) y disgusto (*disgust*)) como el **sentimiento general** de la noticia, clasificado como positivo, negativo o neutral.

Este análisis permite asignar, para cada noticia, un **puntaje** correspondiente a cada una de las seis emociones y a los tres posibles sentimientos generales. Por ejemplo, una noticia podría obtener un 15 % de puntuación positiva, un 15 % negativa y un 70 % neutral, y de manera similar para las emociones. Posteriormente, se calculó un **promedio de estos puntajes** para todo el conjunto de noticias verdaderas y, de manera independiente, para el conjunto de noticias falsas, con el objetivo de comparar ambos tipos de noticias y determinar de manera rápida si esta información aporta valor diferenciador para la clasificación.

Adicionalmente, se identificó para cada noticia la **emoción**, es decir, aquel que presenta el mayor porcentaje, lo que permite un análisis más intuitivo de las características emocionales de las noticias verdaderas y falsas.

Es decir, en primer lugar se calcula, para cada noticia, la **proporción que contiene de cada emoción y de cada sentimiento**. A continuación, para cada noticia se identifica cuál es la emoción y cuál es el sentimiento con **mayor peso relativo**, considerándose estos como los predominantes. Finalmente, para realizar la comparación se emplea la métrica mostrada en la **Figura 27**. En ella se presentan, por un lado, el **promedio de aparición de cada emoción y cada sentimiento en el conjunto completo del corpus** y, por otro, la **proporción de veces que cada emoción y cada sentimiento aparecen como predominantes en una noticia**.

```
--- NOTA FINAL DEL CORPUS HEADLINE TRUE(Promedio de Probabilidades) ---
NEU         43.569220
NEG         42.261246
POS         14.169534
joy          2.888239
anger        2.712343
sadness      2.125259
surprise     1.890562
fear         1.136445
disgust      0.973688
dtype: float64

--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---
emocion_final
others      93.225806
anger        2.419355
joy          2.096774
sadness      1.612903
fear         0.322581
surprise     0.161290
disgust      0.161290
Name: proportion, dtype: float64
```

Figura 27: Emociones y sentimientos de los titulares true

Fuente: Elaboración propia

```
--- NOTA FINAL DEL CORPUS headline fake(Promedio de Probabilidades) ---
NEG         47.974651
NEU         40.536226
POS         11.489122
anger        4.646933
surprise     2.163067
disgust      1.828525
sadness      1.765401
joy          1.720431
fear         1.249687
dtype: float64

--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---
emocion_final
others      92.238267
anger        4.512635
joy          1.083032
sadness      0.722022
surprise     0.722022
disgust      0.541516
fear         0.180505
Name: proportion, dtype: float64
```

Figura 28: Emociones y sentimientos de los titulares fake

Fuente: Elaboración propia

```
--- NOTA FINAL DEL CORPUS TRUE (Promedio de Probabilidades) ---
NEG         44.371923
NEU         40.006664
POS         15.621413
anger        9.933687
sadness      7.779720
joy          3.707440
disgust      1.525027
fear         0.725783
surprise     0.706295
dtype: float64

--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---
emocion_final
others     81.612903
anger       9.838710
sadness     5.322581
joy         3.064516
fear        0.161290
Name: proportion, dtype: float64
```

Figura 29: Emociones y sentimientos de las noticias true

Fuente: Elaboración propia

```
--- NOTA FINAL DEL CORPUS FAKE (Promedio de Probabilidades) ---
NEG         55.267566
NEU         32.650702
anger       19.859573
POS         12.081731
sadness      8.792720
disgust      2.705496
joy          2.084862
surprise     0.991096
fear         0.941633
dtype: float64

--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---
emocion_final
others      70.938628
anger       21.480144
sadness      5.595668
joy          1.444043
fear         0.361011
surprise     0.180505
Name: proportion, dtype: float64
```

Figura 30: Emociones y sentimientos de las noticias fake

Fuente: Elaboración propia

Como se esperaba, las noticias falsas tienden a ser ligeramente menos objetivas que las noticias verdaderas, mostrando una mayor inclinación hacia emociones y sentimientos explícitos. Al analizar los **titulares**, se observa que las **noticias verdaderas presentan mayoritariamente un sentimiento neutral**, y la emoción predominante se clasifica como "otra" en el 93 % de los casos, dado que las seis emociones definidas aparecen como predominantes en un porcentaje muy bajo. En contraste, **las noticias falsas muestran mayoritariamente un sentimiento negativo**, y las seis emociones básicas tienen una frecuencia más elevada como predominantes, quedando la categoría "otra" en un 92 % de los casos. Esto tiene sentido, ya que las noticias falsas suelen buscar captar la atención de los lectores mediante titulares llamativos, con el objetivo de inducirles a acceder a la noticia y aumentar su difusión.

Si se consideran los valores promedio de las seis emociones en los titulares, se observa que **en las noticias falsas la ira,** que es la emoción predominante, alcanza **un 4,6 %,** mientras que en **las noticias verdaderas, la alegría**, que es la emoción predominante, aparece con un **2,88 %,** lo que indica que, en promedio, los titulares de las noticias verdaderas están más equitativos en cuanto a la cantidad de cada emoción que presentan y ademas, pueden presentar un tono más positivo.

En cuanto al **contenido de los textos completos**, **tanto las noticias verdaderas como las falsas tienden hacia un sentimiento negativo, aunque las verdaderas presentan aproximadamente un 10 % menos de negatividad que las falsas. Asimismo, las noticias falsas muestran un mayor porcentaje de predominancia de las seis emociones básicas**. Por ejemplo, la ira aparece como emoción predominante en el 20 % de las noticias falsas, frente a un 9 % en las noticias verdaderas, y el porcentaje promedio de ira en los textos es del 19 % en las noticias falsas frente al 9 % en las verdaderas. Esto refleja que, al igual que en los titulares, las noticias falsas presentan emociones más intensas que las verdaderas.

Estos resultados son coherentes con la hipótesis de que los autores de noticias falsas suelen tener menor profesionalidad o experiencia en la redacción informativa, lo que dificulta mantener un tono objetivo, además, buscan captar la atención del lector y generar emociones en él con el fin de que comparta y difunda el contenido. No obstante, las emociones con mayor porcentaje son similares en ambos tipos de noticias, siendo la **ira** la predominante en casi todas las situaciones. De los análisis realizados, se observa que

las noticias falsas tienden a contener una mayor proporción de emociones negativas, especialmente de **ira**, lo que puede fomentar el **odio**, en comparación con las noticias verdaderas o positivas.

Dado que estos datos proporcionan información potencialmente útil para la clasificación de noticias, se decidió incluirlos como variables en los conjuntos de datos que se introducirán en los modelos de clasificación.

3.3.4 Análisis de los metadatos

Finalmente, se llevó a cabo un análisis de los **metadatos textuales** para determinar cuál de los dos tipos de noticias presenta una mayor densidad de ciertos elementos, tales como signos de exclamación, signos de interrogación, números, longitud de las palabras, cantidad de palabras y uso de mayúsculas. El objetivo de este estudio es caracterizar el **estilo de escritura** asociado a cada categoría (verdadera o falsa) y evaluar si existen diferencias significativas que puedan ser útiles para la construcción de los modelos de clasificación y para comprender mejor las particularidades de cada tipo de noticia.

En concreto, a partir de este análisis se obtuvieron las siguientes métricas:

```
Metadatos extraidos de los textos:
      longitud_char  n_palabras  ratio_mayusculas  n_exclamaciones  \
1052            757         120          0.031667                0
191            2060         329          0.019631                0
1288           3538         558          0.031073                0
963            1672         281          0.032975                0
1380           8740        1459          0.032512                0

      n_interrogaciones  n_numeros  avg_word_length
1052                  0          1         6.256198
191                   0          0         6.242424
1288                  0         12         6.329159
963                   0          0         5.929078
1380                  1         15         5.986301
```

Figura 31: Metadatos

Fuente: Elaboración propia

Una vez obtenidos los metadatos, se procedió a visualizar las diferencias entre las noticias verdaderas y las falsas. Se observó que las noticias verdaderas presentan **mayor cantidad de números, palabras, caracteres y longitud de palabras** en comparación con las noticias falsas, las cuales tendrán un mayor número de **signos de exclamación,** Sin embargo, el ratio de mayúsculas es bastante parecido.

Estos resultados son coherentes con lo planteado anteriormente: las noticias verdaderas tienden a mantener un tono objetivo y a minimizar la expresión de emociones o sentimientos intensos, mientras que las noticias falsas, al carecer de profesionalidad o experiencia, muestran un mayor número de exclamaciones e interrogaciones. Además, las noticias verdaderas, al contar con mayor rigor y profesionalidad, suelen incluir más argumentos y datos, lo que se refleja en textos más extensos, con mayor presencia de cifras y números, y en palabras de mayor longitud promedio. Esto también puede indicar el uso de un **lenguaje más complejo y elaborado** en las noticias verdaderas en comparación con el lenguaje más simple empleado en noticias falsas publicadas en medios menos profesionales.

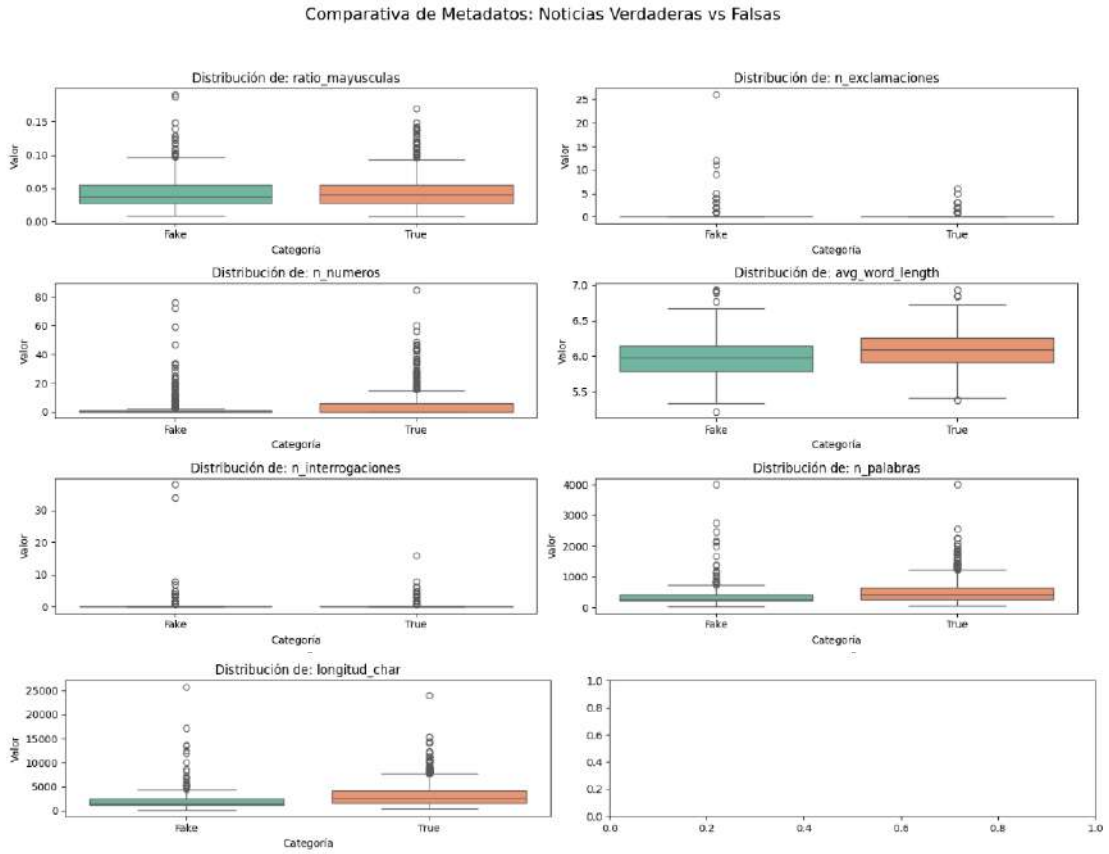


Figura 32: Comparación de los metadatos entre las noticias true y las fake

Fuente: Elaboración propia

```
--- VALORES MEDIOS POR CATEGORÍA ---
```

Category	longitud_char	n_palabras	ratio_mayusculas	n_exclamaciones	n_interrogaciones	n_numeros	avg_word_length
Fake	2124.196751	353.341155	0.043597	0.312274	0.503610	2.541516	5.969836
True	3277.595161	536.714516	0.045260	0.124194	0.427419	4.972581	6.091891

Figura 33: Valores medios de los metadatos de las noticias

Fuente: Elaboración propia

En cuanto a los **títulos de las noticias**, se observa que las noticias falsas presentan un **mayor número de palabras** en comparación con las verdaderas, aunque la longitud promedio de las palabras es similar en ambos casos. Esta diferencia puede explicarse por el hecho de que los redactores profesionales suelen elaborar títulos **breves, concisos y eficaces**, mientras que los autores de noticias falsas, además de carecer de la misma profesionalidad, tienden a crear títulos más extensos con el objetivo de generar un **"gancho"** que atraiga al lector hacia el contenido completo de la noticia.

--- VALORES MEDIOS POR CATEGORÍA ---

Category	longitud_char_h	n_palabras_h	ratio_mayusculas_h	n_exclamaciones_h	n_interrogaciones_h	n_numeros_h	avg_word_length_h
Fake	76.469314	12.666065	0.200665	0.028881	0.016245	0.146209	5.605787
True	67.772581	11.154839	0.086583	0.009677	0.062903	0.170968	5.620269

Figura 34: Valores medios de los metadatos de los titulares

Fuente: Elaboración propia

Los titulares de las noticias falsas también disponen de un mayor ratio de mayúsculas, indicando que pueden estar constituidas por más de una frase y mayor número de exclamaciones, aunque menor número de interrogaciones y de números. Por ello, se considera que los **metadatos textuales** pueden constituir una característica relevante para incluir en los modelos de clasificación, contribuyendo a la diferenciación entre noticias falsas y verdaderas.

Los resultados del análisis indican que, en general, las noticias verdaderas presentan un **estilo de escritura más complejo** que las noticias falsas, caracterizado por un mayor número de palabras (excepto en los titulares), caracteres y cifras, mientras que muestran un comportamiento más **neutral**, con menor presencia de signos de exclamación e interrogación. Por el contrario, las noticias falsas tienden a ser más simples en su estructura y a utilizar estos signos con mayor frecuencia, lo que refuerza la utilidad de los metadatos como variable complementaria en los modelos de clasificación.

En conjunto, **estos resultados coinciden con lo observado en estudios previos, especialmente Horne y Adalı (2017)**, quienes también identifican que las noticias falsas presentan un estilo más simple, emocional y menos elaborado, mientras que las verdaderas muestran mayor complejidad y densidad informativa. Asimismo, la mayor

longitud y carga llamativa de los titulares falsos que se aprecia en este trabajo refuerza la idea de que este tipo de contenido busca captar la atención del lector desde el título, tal como señala la literatura.

3.3.5 Conclusiones del EDA

El análisis exploratorio de los datos ha permitido no solo comprender mejor las características de las noticias falsas, sino también identificar cuáles podrían ser **variables relevantes** (las emociones y sentimientos, los metadatos y la fuente) para introducir en los modelos de clasificación con el objetivo de determinar la veracidad de una noticia. Por ello, una vez finalizado este análisis, se procede a la **preparación de los datos** para su posterior uso en el entrenamiento de los modelos.

Los hallazgos derivados de este análisis exploratorio muestran una notable coherencia con el informe del Reuters Institute, que identifica un creciente temor social ante la desinformación. Esta preocupación queda respaldada empíricamente en el presente estudio, al observarse que las noticias falsas tienden a instrumentalizar emociones de alta intensidad, como la ira, con el objetivo de maximizar la interacción de los usuarios y ampliar, así, el alcance de dicha información falsa.

No obstante, **frente al 58 % de la población encuestada por Reuters que manifiesta inseguridad en la detección de bulos, los datos de este análisis sugieren que dicha percepción de riesgo podría estar sobredimensionada**. Al examinar la muestra, se evidencia que la gran mayoría de las fuentes mantiene una línea editorial consistente, siendo enteramente veraces o completamente falsas, y que la incidencia de noticias falsas y verdaderas procedentes de una misma fuente es reducida, con apenas 11 de 296 casos identificados. Estos resultados sugieren que la detección de desinformación podría no ser tan compleja como se percibe, ya que, en este conjunto de datos, basta en muchos casos con identificar la fuente para distinguir entre contenidos veraces y falsos.

3.4 Modelo basado en embeddings

En este apartado se implementaron distintos modelos de clasificación aprendidos en clase, seleccionando posteriormente aquel que presentó un mejor desempeño en la clasificación de noticias, con el objetivo de compararlo más adelante con un **LLM**. Los modelos

evaluados fueron **Random Forest**, **Logistic Regression**, **XGBoost y SVM**, probándose cada uno con diferentes variaciones.

Previo al entrenamiento de estos modelos, fue necesario generar **embeddings** a partir de las noticias y seleccionar y preparar las variables, ya que estos modelos no comprenden de forma nativa el significado de las palabras, el contexto o el estilo de escritura, sino que requieren datos numéricos que representen dicha información.

3.4.1 Selección de variables

Introducir un exceso de información en los modelos puede conducir a un fenómeno conocido como **overfitting**. Según Schaffer (1993), el overfitting se produce cuando un modelo complejo alcanza una alta precisión en el conjunto de datos de entrenamiento, pero posteriormente presenta un desempeño inferior en datos nuevos en comparación con un modelo más simple, debido a que se ha ajustado excesivamente a los patrones específicos del conjunto de entrenamiento, incluyendo aquellos determinados por azar, en lugar de reflejar únicamente patrones generales y significativos.

Dado que la probabilidad de incurrir en overfitting aumenta con la complejidad del modelo, **el análisis exploratorio realizado permitió identificar qué variables aportan valor real a la clasificación y cuáles no**. En este sentido, se concluyó que información como el topic no resulta útil para la clasificación, ya que únicamente incrementaría la complejidad del modelo sin mejorar su capacidad predictiva.

Ademas, en el caso de los metadatos, se observó que algunas variables presentan una **correlación muy alta**, particularmente entre la **longitud total de caracteres** y el **número de palabras** de la noticia. Esto resulta lógico, ya que, a mayor número de palabras, mayor será el total de caracteres. Para evitar un aumento innecesario de la complejidad del modelo y el riesgo de **overfitting**, se decidió eliminar una de estas variables. Entre la longitud máxima de caracteres y el número de palabras, se optó por eliminar la **longitud máxima de caracteres**, ya que los metadatos incluyen también la **longitud media de las palabras**, la cual, combinada con el número de palabras, proporciona información suficiente sobre el tamaño y la extensión de la noticia.

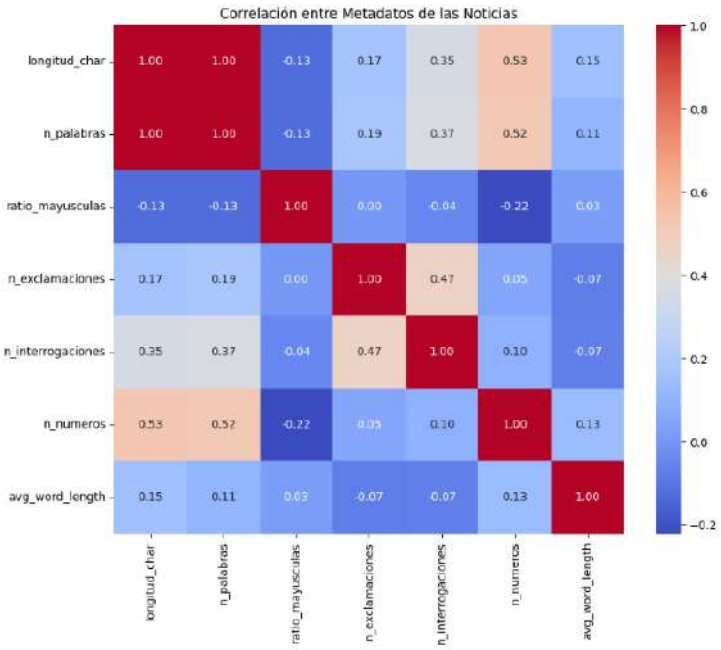


Figura 35: Correlación entre los metadatos de las noticias

Fuente: Elaboración propia

Además, para reducir la correlación entre variables y minimizar el riesgo de **overfitting**, se eliminaron las variables **emoción final** y **polaridad final**, ya que actúan como un resumen de la información contenida en el resto de las variables, indicando únicamente cuál es la emoción o el sentimiento predominante en cada noticia, es decir, la que presenta un mayor porcentaje.

A partir de este análisis exploratorio, se construyó un **nuevo conjunto de datos** que incluye toda la información relevante de la base original, los metadatos, las emociones y sentimientos y eliminado variables como el **topic**, la longitud de caracteres y las variables de sentimiento y emoción finales.

Para poder entrenar los modelos de clasificación, los datos debieron ser previamente preparados, codificados y normalizados cuando fue necesario. Por ejemplo, la variable **Category** se transformó a formato numérico, asignando 0 a noticias falsas y 1 a noticias verdaderas.

3.4.2 Embeddings

Mars (2022) explica que los **embeddings** son representaciones de palabras o fragmentos de texto en forma de **vectores numéricos densos** dentro de un espacio vectorial de

dimensiones específicas. Estas representaciones permiten que los modelos de aprendizaje automático puedan procesar información textual al transformarla en valores numéricos que capturan relaciones semánticas entre los distintos elementos del lenguaje.

En este trabajo se ha utilizado **MiniLM**, un modelo más avanzado que los métodos iniciales de generación de embeddings y que constituye una optimización de la arquitectura de **BERT** mediante técnicas de **destilación del conocimiento**. Aunque se trata de un modelo más eficiente que BERT en términos de tamaño y coste computacional, no alcanza el nivel de complejidad de un **LLM (Large Language Model)**, suele clasificarse dentro de los **PLM (Pre-trained Language Models)**.

Este tipo de modelo pre-entrenado genera **embeddings contextuales a nivel de oración**, es decir, no representa únicamente palabras individuales, sino frases completas, teniendo en cuenta el contexto en el que aparecen. De esta forma, frases que expresan una idea similar utilizando palabras diferentes se ubican en zonas cercanas del espacio vectorial, lo que permite capturar relaciones semánticas entre ellas.

La elección de este modelo se debe principalmente a su **menor tamaño y coste computacional** en comparación con modelos más grandes, manteniendo al mismo tiempo un buen rendimiento. MiniLM aprende a **imitar el comportamiento de modelos de mayor tamaño** mediante técnicas de destilación. (Mars, 2022).

Tal y como se mencionó previamente en la descripción técnica de los transformers, este modelo pertenece a la familia de arquitecturas **encoder-only**, es decir, está compuesto únicamente por un encoder y no dispone de un decoder. Su función principal consiste en **leer y comprender el texto**, transformando el contenido del artículo en un vector numérico capaz de representar su significado. Gracias a su naturaleza destilada, MiniLM permite disponer de un modelo **más pequeño y rápido**, manteniendo una buena capacidad para capturar **similitudes semánticas entre frases**.

Este enfoque no constituye la forma más tradicional de detectar noticias falsas mediante técnicas de machine learning. Sin embargo, sí representa una de las aproximaciones que comenzó a utilizarse con mayor frecuencia en el ámbito del **Procesamiento del Lenguaje Natural (NLP)** para capturar el significado semántico del texto.

Asimismo, se ha desarrollado un modelo de detección de noticias falsas basado en el algoritmo Random Forest, en el cual se incorporan como variables de entrada unicamente la clasificación de la noticia como su contenido representado mediante embeddings. Los resultados obtenidos confirman la hipótesis planteada por Ma et al. (2024), evidenciando que el uso exclusivo de embeddings no resulta suficientemente eficaz para la identificación de noticias falsas. Esto se debe a que este tipo de contenido suele imitar el estilo lingüístico de las noticias verídicas, lo que dificulta su discriminación por parte de los modelos de aprendizaje automático.

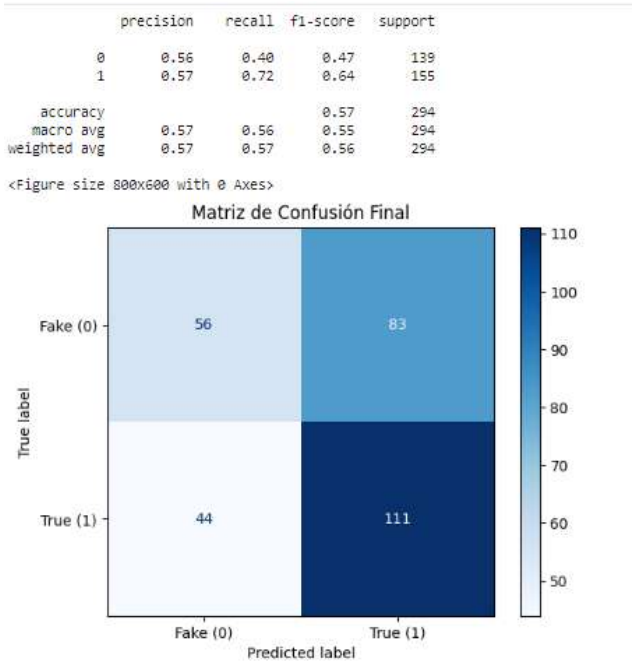


Figura 36: Resultados obtenidos en el modelo de clasificación realizado únicamente con embeddings

Fuente: Elaboración propia

Es por ello por lo que, para poder construir modelos de detección de notificas falsas, fue necesario **generar los embeddings** de las noticias y concatenarlos con el resto de las variables seleccionadas previamente para el modelo de clasificación. Posteriormente, se definió como variable objetivo (y) la **categoría de la noticia** (true o fake). Además, se aplicó un proceso de **escalado de los datos**, con el fin de evitar que variables con magnitudes grandes dominen el modelo.

A continuación, se aplicaron las mismas funciones utilizadas en el conjunto de **entrenamiento** para calcular los **sentimientos, emociones y metadatos** sobre el conjunto de **test**, asegurando así que ambos conjuntos de datos presentasen la misma estructura y

características. Este paso resulta especialmente importante, ya que el conjunto de test debe transformarse siguiendo exactamente el mismo procedimiento que el conjunto de entrenamiento.

Asimismo, es fundamental que las variables del conjunto de test se encuentren **en el mismo orden, con las mismas transformaciones y bajo la misma escala** que las del conjunto de entrenamiento, de modo que el modelo pueda interpretarlas correctamente durante el proceso de evaluación.

Una vez finalizado el proceso de preparación de los datos, se procedió a implementar diversos modelos de clasificación, así como distintas variantes de cada uno de ellos, con el objetivo de evaluar de forma comparativa su rendimiento en la detección de noticias falsas. De este modo, se pretende evitar extraer conclusiones basadas en el comportamiento de un único modelo. En línea con lo expuesto previamente, el propósito principal es analizar si los enfoques más recientes ofrecen realmente mejoras respecto a modelos anteriores. Para ello, se adopta una estrategia experimental basada en la evaluación de múltiples modelos, minimizando así el riesgo de que los resultados estén condicionados por la elección de un clasificador concreto.

3.4.3 Random forest

En primer lugar, se implementó un modelo de **Random Forest** utilizando el conjunto completo de variables previamente escaladas. Posteriormente, se evaluó una segunda variante del mismo modelo en la que se eliminaron las variables con **altos niveles de correlación**, concretamente aquellas con una correlación superior a **0.6**, eliminando únicamente una variable de cada par correlacionado. El objetivo de esta modificación fue reducir la redundancia entre variables, simplificar el modelo y minimizar el riesgo de **overfitting** mencionado anteriormente.

Dado que esta segunda versión del modelo mostró una mejora en el rendimiento respecto al modelo inicial, se probó una tercera variante en la que se realizó una **selección de características**, conservando únicamente las **35 variables más relevantes**. Sin embargo, este enfoque resultó demasiado restrictivo, ya que reducía significativamente la cantidad de información disponible para el modelo. En particular, al limitar el número de variables, se descartaba una parte importante de los **embeddings**, lo que implicaba perder información semántica del texto.

Dado que el último modelo basado en la selección de únicamente 35 variables podía resultar demasiado restrictivo y provocar una pérdida significativa de información, se exploró una alternativa basada en la **reducción de dimensionalidad**. En concreto, se aplicó **Análisis de Componentes Principales (PCA)** sobre los embeddings con el objetivo de reducir la complejidad, dimensionalidad y ruido del modelo, disminuir el riesgo de **overfitting** y mantener, al mismo tiempo, la mayor cantidad posible de información relevante.

Mediante este proceso se redujo la dimensionalidad de los embeddings, pasando de **383 variables originales** que representaban la información textual a **49 componentes principales**, lo que supone una reducción considerable del número de dimensiones del espacio vectorial.

A continuación, se volvió a entrenar un modelo de **Random Forest** utilizando estos embeddings transformados mediante PCA. Al igual que en los experimentos anteriores, se evaluaron distintas variantes del modelo, incluyendo versiones en las que se eliminaron variables con alta correlación.

La mayoría de los estudios en el ámbito de la detección de noticias falsas, como el desarrollado por William Yang Wang (2017), **evalúan el rendimiento de los modelos de clasificación mediante métricas agregadas como la exactitud global (accuracy) o el F1-score.** Este enfoque resulta adecuado como primera aproximación, ya que permite comparar de manera objetiva distintos modelos y determinar cuáles presentan un mejor desempeño general.

No obstante, en este tipo de problemas**, limitar el análisis a estas métricas puede resultar insuficiente**. En particular, adquiere relevancia la necesidad de reducir el número de falsos positivos dentro del proceso de clasificación.

Por ello, en el presente Trabajo de Fin de Grado, además de la evaluación de los modelos principales, se llevaron a cabo experimentos adicionales centrados en **el ajuste del umbral de clasificación** empleado para determinar si una noticia debe ser considerada falsa o verdadera, con el objetivo de analizar su efecto sobre las distintas métricas de rendimiento.

El objetivo de esta estrategia fue seleccionar un modelo que, aunque pudiera mostrar una **precisión ligeramente inferior**, redujera la cantidad de **noticias falsas no detectadas**,

priorizando así la capacidad del modelo para identificar correctamente las noticias falsas en lugar de maximizar únicamente la accuracy global.

En la sociedad actual, el principal objetivo de estos modelos es **detectar la mayor cantidad posible de información falsa**, evitando que la población sea inducida a creer noticias incorrectas. La difusión de información falsa no solo afecta al conocimiento y la cultura general, sino que puede tener repercusiones significativas sobre los gobiernos, la economía de los países, los resultados de las empresas e incluso la reputación de individuos, lo que subraya la importancia de su detección.

Por este motivo, el objetivo de los modelos es, contando con un sistema **mínimamente robusto**, **maximizar la detección de noticias falsas** mientras se mantiene el número de falsos positivos (noticias predichas como verdaderas pero que en realidad son falsas) lo más bajo posible. Con este fin, se realizaron experimentos ajustando ligeramente el **umbral de decisión**, buscando un equilibrio que permitiera reducir los falsos positivos sin degradar significativamente el desempeño general del modelo. Por ejemplo, al aumentar el umbral de decisión en el modelo Random Forest, tras eliminar las variables correlacionadas, el número de falsos positivos se reduce de 44 a 27.

En términos de métricas, esto equivale a priorizar un **recall elevado**, es decir, garantizar que el modelo sea capaz de identificar la mayor proporción posible de noticias falsas presentes en el conjunto de datos.

De todos los modelos de **Random Forest** evaluados, se puede confirmar que la versión que incorpora **PCA** presenta un mejor desempeño general. Este modelo no solo mantiene una **accuracy adecuada**, sino que también logra uno de los valores más bajos de **falsos positivos (FP)**. Además, en esta variante se combinó la reducción de dimensionalidad mediante PCA con la **eliminación de variables altamente correlacionadas**, lo que contribuyó a simplificar el modelo sin sacrificar información relevante y a mejorar su capacidad para identificar noticias falsas de manera efectiva.

3.4.4 Logistic regression y random forest

A continuación, se evaluó un **modelo híbrido**, en el que se emplea **Logistic Regression (LR)** para los embeddings y **Random Forest (RF)** para el resto de las variables, incluyendo metadatos, emociones y la fuente de la noticia. Esta combinación cuenta con

una base metodológica sólida según el estudio comparativo presentado en Random Forest versus Logistic Regression de Couronné, Probst y Boulesteix (2018).

El artículo indica que **LR funciona especialmente bien cuando los datos presentan relaciones aproximadamente lineales** y la dimensionalidad no es excesiva. En nuestro caso, los embeddings constituyen un espacio vectorial **continuo y denso**, donde las relaciones semánticas pueden aproximarse linealmente, cumpliendo con las condiciones óptimas de LR. Tal como se señala en el estudio: *"*the prediction performance of LR depends on whether the data follow the assumed model*"* [El rendimiento de predicción de la regresión logística depende de si los datos siguen el modelo supuesto], y LR es el enfoque estándar en contextos relativamente simples y estructurados.

Por otro lado, **LR no es adecuado cuando el número de variables es muy elevado o cuando existen interacciones complejas entre ellas**. El artículo destaca que: "RF captures the dependence and non-linearity structures… logistic regression is not able to." [RF captura las estructuras de dependencia y no linealidad… algo que la regresión logística no es capaz de hacer]. Esto coincide con la situación de nuestros metadatos, emociones y especialmente los cientos de variables categóricas asociadas a la fuente, donde las relaciones no son lineales y dependen de combinaciones específicas de variables. Por ejemplo, un alto número de exclamaciones puede no aportar información por sí solo, pero combinado con una fuente determinada puede constituir un indicador relevante. **RF es capaz de detectar automáticamente estos patrones**, mientras que LR requeriría la introducción manual de todas las posibles interacciones.

Por tanto, el uso exclusivo de LR habría dificultado capturar la complejidad y no linealidad de los metadatos, mientras que emplear únicamente RF habría desaprovechado la estructura más lineal y continua presente en los embeddings. La combinación de ambos modelos permite **aprovechar las fortalezas de cada uno según el tipo de información que procesan**.

Al igual que con los otros modelos, se evaluaron distintas variantes de este enfoque híbrido, probando tanto la aplicación de **PCA sobre los embeddings** como la **selección de las variables más relevantes** o la **eliminación de aquellas con alta correlación**. Dado que LR funciona mejor con un número reducido de variables, la configuración que obtuvo mejor desempeño fue aquella que combinaba **PCA en los embeddings** y

eliminación de variables altamente correlacionadas en los metadatos, maximizando así la eficiencia y la capacidad predictiva del modelo.

Este método constituye una de las contribuciones metodológicas del presente Trabajo de Fin de Grado, al proponer el desarrollo de un enfoque híbrido de clasificación. A diferencia de la literatura previa, como el trabajo de Posadas-Durán et al. (2019), en el que los modelos se aplican de manera aislada sobre representaciones vectoriales de palabras, en esta investigación se plantea una integración estratégica de distintos algoritmos con el fin de explotar sus fortalezas complementarias.

Esta combinación permite abordar de manera más eficaz la naturaleza heterogénea de los datos, integrando tanto información semántica como estructural en un único marco de análisis. Como resultado, la arquitectura propuesta logra aprovechar las ventajas de ambos enfoques, lo que se traduce en una mayor robustez del modelo frente a métodos de clasificación basados en un único algoritmo. Asimismo, esta técnica presenta un rendimiento notable, alcanzando una precisión (accuracy) del 86 %, lo que la sitúa entre los modelos manuales más competitivos analizados en el presente TFG.

3.4.5 XGBoost

Se evaluó un tercer modelo, **XGBoost**, un enfoque más avanzado dentro de los métodos de **boosting**. Según Bentéjac, Csorgo y Martínez Muñoz (2019), este tipo de modelos se caracteriza por su **alta precisión** y por contar con un **algoritmo sensible a la dispersión**, lo que resulta especialmente útil en bases de datos con un gran número de variables de diferentes tipos, como es nuestro caso. Además, XGBoost incorpora mecanismos de **regularización** que penalizan el sobreajuste (overfitting), lo que permite que el modelo generalice mejor y no se limite a memorizar los datos de entrenamiento, favoreciendo un desempeño más robusto frente a nuevos conjuntos de datos. Para dicho modelo, igual que con los anteriores, también se han evaluado distintas variantes.

3.4.6 SVM

Finalmente, se evaluó un último modelo: la Support Vector Machine (SVM). Este modelo se seleccionó por su capacidad para identificar un hiperplano óptimo que separe linealmente las dos clases, “fake” y “true”, lo que lo convierte **en un clasificador especialmente relevante frente a otros enfoques (**Abdullah & Abdulazeez, 2021).

Para su evaluación, se entrenó inicialmente el modelo con los datos originales, y posteriormente se exploraron distintas variantes, incluyendo el uso de embeddings reducidos mediante PCA y la eliminación de características altamente correlacionadas. Esta estrategia permitió analizar cómo distintos preprocesamientos de los datos afectan el rendimiento de la SVM en la detección de noticias falsas.

3.4.7 Selección del mejor modelo

Entre todos los modelos evaluados, los cuatro que presentan mejor desempeño son: (1) el **modelo híbrido**, que combina **Logistic Regression sobre los embeddings** con **Random Forest** aplicado a los metadatos tras eliminar variables altamente correlacionadas, (2) el **Random Forest con PCA aplicado a los embeddings** tras eliminar variables altamente correlacionadas y subir el umbral, (3) el XGBoost con PCA aplicado sobre los embeddings y (4) el SVM habiendo quitado las variables con mayor correlación y subiendo el umbral de clasificación. La principal diferencia entre ellos radica en que el segundo modelo logra una **mejor clasificación de noticias falsas** y una reducción de los **falsos positivos**, aspecto que representa nuestro objetivo principal de optimización, mientras que el primer modelo presenta un **desempeño global más elevado** en términos generales y el tercer y cuarto modelo son una combinación de ambos donde se obtienen unos buenos niveles de falsos positivos y unos buenos desempeños globales pero sin ser el mejor en ninguno de los dos campos.

Por ello, se decidió tomar ambos modelos para realizar un **proceso de optimización adicional**, con el objetivo de determinar cuál produce mejores resultados y cuál será finalmente seleccionado para su **comparación con el LLM**. Para dicha optimización, se ha ajustado el umbral de clasificación con el objetivo de **maximizar la precisión** (Precisión = {TP}/{TP + FP}), minimizando así el impacto de los **falsos positivos**. Esta búsqueda se ha realizado bajo una **restricción de control**, asegurando que la **exactitud global (accuracy)** no descienda del **80%**, lo que garantiza que el modelo mantenga un equilibrio robusto entre la fiabilidad de sus predicciones positivas y su capacidad predictiva general.

	precision	recall	f1-score	support
0	0.88	0.81	0.84	139
1	0.84	0.90	0.87	155
accuracy			0.86	294
macro avg	0.86	0.85	0.86	294
weighted avg	0.86	0.86	0.86	294

<Figure size 800x600 with 0 Axes>

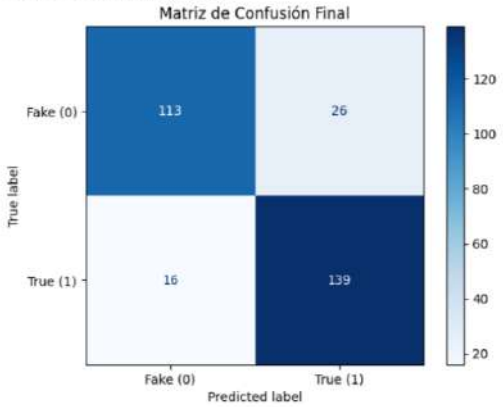


Figura 37: Resultados modelo de clasificación logistic regression + random forest

Fuente: Elaboración propia

	precision	recall	f1-score	support
0	0.78	0.86	0.82	139
1	0.86	0.78	0.82	155
accuracy			0.82	294
macro avg	0.82	0.82	0.82	294
weighted avg	0.82	0.82	0.82	294

<Figure size 800x600 with 0 Axes>

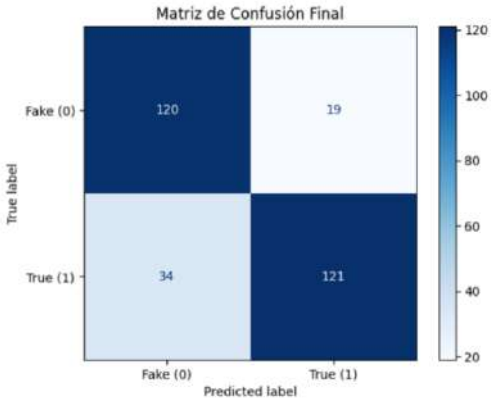


Figura 38: Resultados modelo de clasificación random forest con PCA sobre los embeddings

Fuente: Elaboración propia

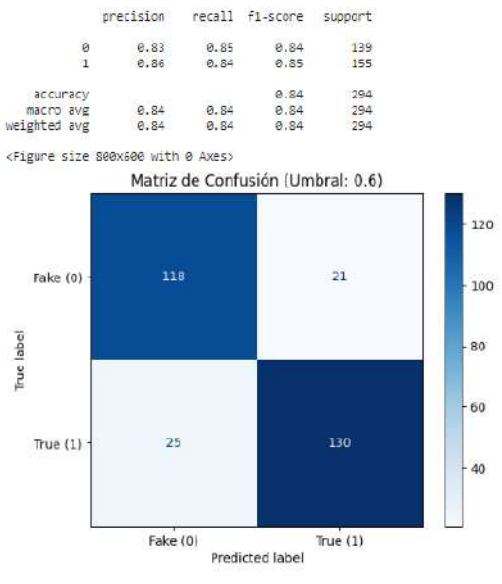


Figura 39: Resultados modelo de clasificación XGBoost

Fuente: Elaboración propia

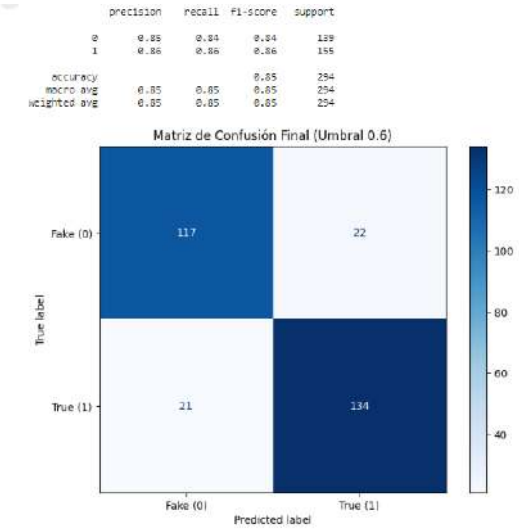


Figura 40: Resultados modelo de clasificación SVM

Fuente: Elaboración propia

Tras realizar la **optimización de los cuatro modelos**, se seleccionó como el más adecuado el **modelo SVM**, quedando reflejados los siguientes resultados:

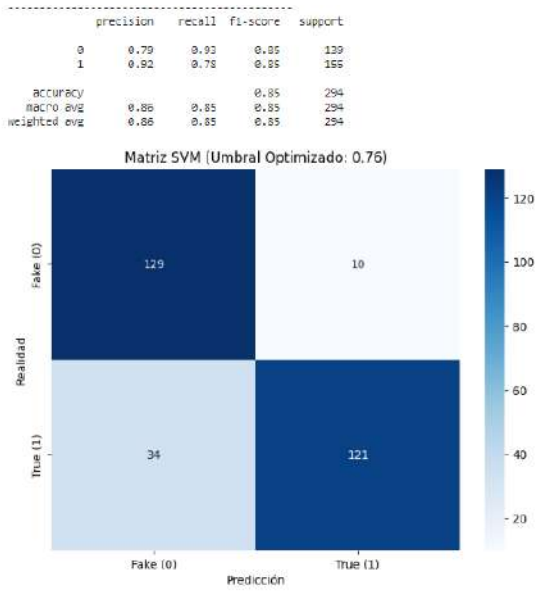


Figura 41: Modelo de clasificación SVM optimizado

Fuente: Elaboración propia

Este no es el modelo que más reduce los falsos positivos; el modelo híbrido, representado en la Figura 42, presenta mejores valores en este aspecto. Sin embargo, si se considera tanto un nivel bajo de falsos positivos como el desempeño global, el SVM es el que ofrece los mejores resultados, seguido por el modelo híbrido.

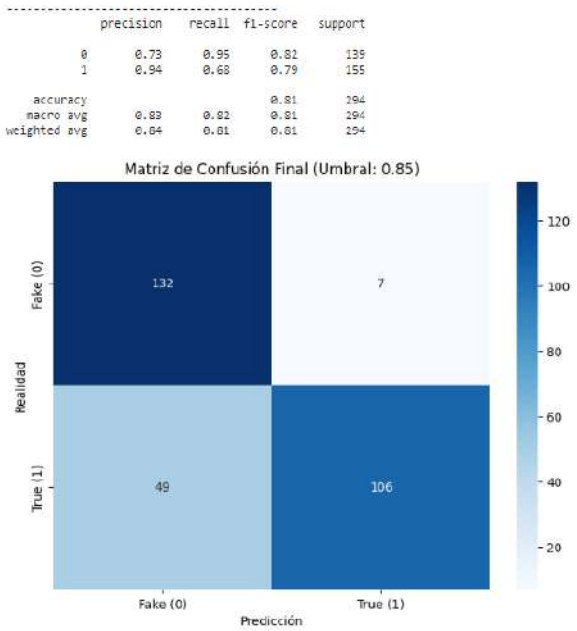


Figura 42: Modelo optimizado logistic regression + random forest

Fuente: Elaboración propia

3.5 LLM

Tras este primer enfoque basado en modelos clásicos de machine learning, el campo del **Procesamiento del Lenguaje Natural (NLP)** ha experimentado una evolución significativa, dando lugar a los actuales **Large Language Models (LLM)**, entre los que destaca uno de los más conocidos, **ChatGPT**. Por este motivo, en este trabajo también se ha entrenado un modelo de características similares, concretamente **Falcon3-1B**.

Según describe Mars (2022), este modelo comparte la **arquitectura tipo decoder propia de la familia GPT**, lo que significa que pertenece a la categoría de modelos **decoder-only**. Este tipo de modelos no solo es capaz de comprender el significado del texto que se le introduce, sino que también puede **generar texto**, ya que su funcionamiento se basa en predecir la **siguiente palabra** o el **token objetivo** en función del contexto previo. Además, estos modelos pueden responder a **instrucciones específicas**, realizar tareas de razonamiento o generar resúmenes del contenido proporcionado.

Aunque **Falcon3-1B** es más pequeño que otros modelos ampliamente conocidos, como **GPT-3**, su tamaño sigue siendo considerablemente mayor que el del modelo utilizado previamente para generar los embeddings (**MiniLM**). La elección de este modelo se debe principalmente a las **limitaciones computacionales disponibles para la realización de este TFG**, ya que modelos de mayor tamaño requieren una capacidad de procesamiento significativamente superior o son de pago. No obstante, Falcon3-1B cuenta con un **mayor número de parámetros** que MiniLM, lo que le permite capturar patrones lingüísticos más complejos y realizar tareas más avanzadas relacionadas con la comprensión y generación de lenguaje natural.

La principal diferencia funcional entre ambos modelos radica en que **MiniLM únicamente transforma el texto en representaciones numéricas (embeddings)** que capturan su significado semántico, mientras que **Falcon3-1B es capaz de interpretar el texto y generar respuestas**, seguir instrucciones o resumir información. En este sentido, su comportamiento se asemeja más al de sistemas conversacionales como ChatGPT, que son utilizados habitualmente en el día a día para interactuar con modelos de lenguaje avanzados.

En este apartado dedicado a los **LLM** también se han evaluado distintas configuraciones del modelo. En primer lugar, se desarrolló un modelo en el que únicamente se utilizaban como entrada el **título** y el **texto** de cada noticia. Posteriormente, se implementó una

segunda versión del modelo en la que también se incorporó la **fuente** de la noticia como variable adicional, ya que, como se observó en el **análisis exploratorio de los datos**, esta característica proporciona información relevante para la clasificación. Los metadatos, así como los sentimientos y las emociones, también se mostraron como variables relevantes; sin embargo, el LLM ya es capaz de extraer esta información directamente a partir de los textos.

Este segundo modelo mostró un rendimiento considerablemente bueno, alcanzando un **alto nivel de accuracy**. No obstante, dada la importancia de **detectar correctamente las noticias falsas** y evitar la difusión de información errónea, se desarrolló una tercera variante del modelo en la que se **optimizo el umbral de decisión**. El objetivo de esta modificación fue **reducir el número de falsos positivos**, es decir, casos en los que una noticia falsa es clasificada como verdadera, manteniendo al mismo tiempo un nivel mínimo aceptable de accuracy.

Para ello, el modelo se configuró de manera que **solo clasifique una noticia como verdadera cuando el nivel de confianza es elevado**. En concreto, el nivel de umbral optimo es del 95% indicando que únicamente clasifica algo como verdadero si esta al 95% seguro de ello. Este enfoque permite disminuir los falsos positivos, aunque implica inevitablemente un incremento en los **falsos negativos**, es decir, aquellas noticias que, siendo verdaderas, son clasificadas como falsas.

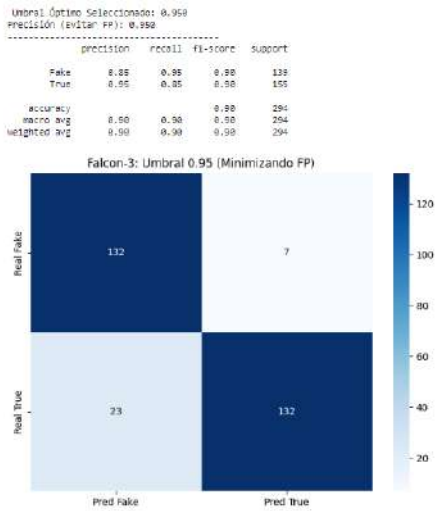


Figura 43: Modelo de clasificación LLM

Fuente: Elaboración propia

4.Resultados

De este modo, el **modelo finalmente seleccionado** es el tercer modelo desarrollado con **Falcon 3-1B**, al ser el que presenta el mejor equilibrio entre las métricas de evaluación consideradas. Es decir, los modelos más "avanzados" muestran un mejor desempeño en la clasificación de noticias falsas.

En este sentido, los hallazgos del presente trabajo **ratifican la tendencia descrita por Jingyuan et al. (2025)**, quien postula que la detección de desinformación ha evolucionado desde la dependencia de la extracción manual de características hacia soluciones automáticas más robustas, representadas en este estudio por la arquitectura **Falcon 3-1B**. Asimismo, **el rendimiento superior de este modelo refuerza lo expuesto por Naveed et al. (2024)** respecto a la capacidad diferencial de los LLM para interpretar matices complejos como el contexto, la ironía y el sarcasmo; dimensiones donde los modelos basados exclusivamente en embeddings mostraron limitaciones en comparación con los LLM.

Los resultados obtenidos por este modelo pueden observarse en la Figura 43.

Este modelo presenta **137 noticias clasificadas como verdaderas que efectivamente lo son** y **131 noticias clasificadas como falsas que también son realmente falsas**. Por otro lado, se producen **18 casos en los que una noticia verdadera es clasificada como falsa** y **únicamente 8 casos en los que una noticia falsa es clasificada como verdadera**, lo que representa un ratio de error reducido y un desempeño muy favorable.

Si se utilizase este modelo para determinar la veracidad de una noticia, sería capaz de **detectar correctamente el 95% de las noticias falsas**, lo que corresponde al **recall de la clase falsa**, es decir, únicamente no se identificarían aproximadamente el **5% de las noticias falsas**.

En cuanto a la **precisión de la clase falsa**, esta alcanza un **85%**, lo que indica que, de todas las noticias que el modelo clasifica como falsas, el **85% efectivamente lo son**. Esta métrica resulta ligeramente inferior al recall, ya que el modelo se ha configurado para priorizar la detección de noticias falsas. En concreto, solo clasifica una noticia como verdadera cuando la probabilidad de que lo sea supera el 95%, bajo la premisa de que el coste social de considerar verdadera una noticia falsa es mayor que el de cometer el error opuesto.

Por su parte, el modelo presenta un recall del 85% en la clase “verdadera”, es decir, **identifica correctamente el 85% de las noticias verdaderas**, ya que, como he mencionado, únicamente clasifica una noticia como verdadera cuando existe un alto nivel de confianza. En consecuencia, **el 95% de las noticias clasificadas como verdaderas realmente lo son**, mostrando un comportamiento consistente y fiable.

De hecho, si observamos la figura 44, se aprecia que la mayoría de los errores del modelo corresponden a noticias que han sido clasificadas como “Fake” cuando en realidad eran “True”. Analizando estos fallos, se puede deducir que en gran parte se deben a un umbral de decisión demasiado elevado. No obstante, también se han producido errores con fuentes como ABC, El Universal o Milenio, donde el modelo ha aprendido que las noticias suelen ser mayoritariamente “true” y, en estos casos concretos, eran “fake”.

	Source	Real	Predicho	Confianza_Score	Distancia_Umbral
24	Salud con Lupa	True	Fake	0.0226	0.9274
19	NoticiaTrabajo	True	Fake	0.0675	0.8825
9	Merca2.0	True	Fake	0.0920	0.8580
25	Xataka	True	Fake	0.1176	0.8324
14	SDP Noticia	True	Fake	0.1520	0.7980
2	Noticia & protagonistas	True	Fake	0.2878	0.6622
15	El Clarin	True	Fake	0.3040	0.6460
20	sputnkinews	True	Fake	0.3124	0.6376
16	Bladi	True	Fake	0.3329	0.6171
12	El Clarin	True	Fake	0.4017	0.5483
5	Dos Mundos	True	Fake	0.4785	0.4715
29	Expansión	True	Fake	0.4805	0.4695
7	Medio Tiempo	True	Fake	0.4863	0.4637
23	Quien	True	Fake	0.4961	0.4539
6	Expansión	True	Fake	0.4980	0.4520
4	Nacion321	True	Fake	0.5794	0.3706
28	Tribuna Noticias	True	Fake	0.7432	0.2068
10	Aristegui noticias	True	Fake	0.8634	0.0866
3	France 24	True	Fake	0.8856	0.0644
22	La Razon	True	Fake	0.8948	0.0552
21	Milenio	Fake	True	0.9998	0.0498
13	La República	Fake	True	0.9992	0.0492
26	El Universal	Fake	True	0.9991	0.0491
18	NEWSner	Fake	True	0.9956	0.0456
8	ABC	Fake	True	0.9868	0.0368
0	El Economista	True	Fake	0.9137	0.0363
27	La Libertad	Fake	True	0.9837	0.0337
11	Prensa Libre	True	Fake	0.9395	0.0105
1	RTVE	True	Fake	0.9399	0.0101
17	Don Diario	Fake	True	0.9588	0.0088

Figura 44: Noticias en las que el modelo se ha equivocado

Fuente: Elaboración propia

Finalmente, el **accuracy**, que refleja el **rendimiento global del modelo**, alcanza un valor del **90%**, lo que indica que **el 90% de todas las clasificaciones realizadas por el modelo son correctas**, un resultado considerado elevado para la tarea de detección de noticias falsas.

Por tanto, se puede concluir que este modelo constituye una **herramienta adecuada y eficaz para la detección de información falsa**.

4.1 LLM Vs Basic

4.1.1 Ventajas

Gracias a este análisis, hemos podido observar **cómo ha evolucionado la detección de noticias falsas y el análisis del lenguaje humano mediante técnicas de machine learning**. Las principales ventajas de estos avances no solo radican en la mejora de los resultados, sino también en que permiten construir modelos **mucho más eficientes y menos laboriosos** que los enfoques tradicionales.

Anteriormente, para que los modelos pudieran capturar correctamente la esencia de una noticia, era necesario realizar un **análisis exhaustivo y un procesamiento manual de los datos**. Por ejemplo, en este trabajo fue necesario analizar las **emociones y los sentimientos** tanto de los títulos como del cuerpo de las noticias, realizar un **análisis de los metadatos**, y transformar las **fuentes** en variables dicotómicas, entre otras tareas, a pesar de tratarse de un modelo relativamente sencillo y con un volumen limitado de información. En escenarios reales, este tipo de análisis puede implicar **un consumo considerable de tiempo y recursos**.

En contraste, los **LLM** son capaces de procesar automáticamente gran parte de esta información. Como se ha mencionado, estos modelos generalmente ya vienen **pre-entrenados**, y aunque se pueden **ajustar o afinar** para tareas específicas, requieren **mucho menos trabajo manual**, lo que simplifica el proceso y disminuye la probabilidad de error humano. Un LLM es capaz de **identificar automáticamente los títulos, el cuerpo del texto, interpretar ironías, reconocer sentimientos y analizar metadatos**, sin necesidad de que estos elementos se introduzcan manualmente, proporcionando así un enfoque más integral y eficiente para la detección de información falsa. Esta mejora en el análisis del texto se aprecia especialmente al comparar los resultados del modelo basado únicamente en embeddings con los del modelo LLM al que se le proporciona únicamente el texto y el título.

Además, si se analiza en detalle, en la mayoría de los casos (salvo en el SVM), los modelos que ofrecieron mejores resultados incluían PCA. Esto puede deberse a que, al tener que introducir las variables de forma manual, estos modelos se enfrentan a un gran número de características, lo que puede provocar **overfitting**. Esta situación se agravará a medida que aumente el número de noticias, dado que se incrementará también la cantidad de embeddings.

4.1.2 Limitaciones

No obstante, estos modelos también presentan **limitaciones y desafíos importantes**. Por un lado, existen **LLM mucho más avanzados** que el utilizado en este TFG, como **GPT** o **LLaMA**, cuyo uso requiere **gran capacidad de procesamiento** (CPU/GPU) y, en muchos casos, acceso de pago, como sucede con la API de GPT. Esto limita el acceso a los LLM más potentes para ciertos usuarios.

Además del **coste computacional y energético**, incluso modelos relativamente sencillos, como el empleado en este TFG, requieren **tiempos de carga y procesamiento superiores** a los de los modelos tradicionales de *machine learning*. Tal como indica Naveed et al. (2024), estas limitaciones técnicas deben considerarse cuidadosamente.

Existen también restricciones propias de su naturaleza: los LLM suelen estar entrenados con **datos estáticos**, por lo que pueden no reflejar información actualizada. Por ejemplo, una de las limitaciones asociadas al uso de **datos estáticos** en este TFG es que algunas fuentes aparecen únicamente una vez en el dataset. Esto puede no ser suficiente para determinar con certeza si la fuente tiende a difundir noticias verdaderas o falsas. Si la única noticia de esa fuente fuese verdadera, el modelo podría aprender que, cuando proviene de dicha fuente, la noticia se tiene que clasificar como verdadera. Aunque este aspecto no constituye un gran limitante, ya que, en la mayoría de los casos, las fuentes tienden a publicar noticias predominantemente verdaderas o falsas, existen fuentes que presentan ambos tipos de noticias, lo que representa una limitación para el modelo. Este problema podría mitigarse si el LLM tuviera acceso a información externa, por ejemplo, mediante búsquedas en bases de datos adicionales.

Es imperativo considerar que otra de las limitaciones asociadas a los datos estáticos son las **limitaciones temporales del entorno de entrenamiento de este modelo**. Los LLMs, incluido el Falcon 3-1B, son sensibles al sesgo presente en los datos de entrenamiento (training data bias). Dado que el corpus utilizado contiene noticias recolectadas entre 2020 y 2021, el modelo ha optimizado sus pesos para identificar patrones de desinformación propios de ese contexto histórico, como la crisis sanitaria global. En consecuencia, el modelo presenta una **limitación de generalización temporal**: ante eventos recientes no representados en el dataset, como conflictos geopolíticos actuales, el sistema podría carecer de referentes semánticos adecuados, aumentando el riesgo de una clasificación incorrecta.

Asimismo, en determinadas situaciones, los modelos pueden generar **"alucinaciones**", es decir, producir respuestas incorrectas o inventadas, lo que podría explicar algunos errores observados en los modelos. Por ejemplo, en nuestro caso, se ha indicado al modelo que solo puede responder con "true" o "fake"; de lo contrario, podría clasificar un caso como "indefinido", por ejemplo.

Otro riesgo es la **privacidad**, ya que los modelos pueden memorizar información sensible introducida por el usuario y, en caso de fallo o ataque, esta información podría ser divulgada.

Otro de los principales problemas es que a pesar de que un LLM no deja de poder ser una "maquina" que te da respuestas no son del todo objetivos porque estan presentes a los sesgos con los que han sido entrenados y, ademas, suelen amplificar los mismos.

Un problema adicional es el **sesgo inherente**. Aunque un LLM sea una máquina, las respuestas que genera no son completamente objetivas, ya que reflejan y a menudo amplifican los **sesgos presentes en sus datos de entrenamiento**. De hecho, como señalan Jingyuan, Zegju, Tianyi y Peiyang (2025), los LLM pueden ser entrenados para **difundir contenido engañoso** si se exponen a sesgos sociales, culturales, económicos o políticos, incluyendo estereotipos, sin que el modelo tenga la capacidad de verificar la veracidad de la información. Esto subraya la **importancia de la transparencia en los datos y de la ética en la creación de modelos**.

En síntesis, los LLM presentan limitaciones claras en términos de **requisitos de hardware**, **coste operacional por tiempos de entrenamiento y procesamiento**, **sesgo**, **falta de privacidad**, **posibles alucinaciones**, y **desactualización de la información**, aspectos que deben considerarse al emplearlos en la detección de noticias falsas.

4.1.3 Mejoras

Sin embargo, como indican Naveed et al (2024), es importante tener en cuenta que el desarrollo de los **LLM apenas ha comenzado**, y aunque presentan ciertos fallos, ya existen **soluciones emergentes** para abordarlos.

Por ejemplo, aunque un modelo relativamente sencillo como el **Falcon 3-1B** utilizado en este TFG tiene **memoria limitada** y no siempre está actualizado, existen LLM **aumentados mediante recuperación de información (Retrieval-Augmented**

Generation, RAG) que pueden acceder a fuentes externas. Esto les permite mantenerse actualizados y ofrecer respuestas más precisas, de manera que incluso modelos más pequeños puedan competir con otros de mayor capacidad y entrenamiento.

En cuanto al **coste computacional y la eficiencia**, también se están desarrollando técnicas para reducirlos. Una de las más conocidas es el **ajuste fino de parámetros (Parameter-Efficient Fine-Tuning, PEFT)**, que adapta el modelo a una tarea concreta de forma eficiente, sin necesidad de modificar todos sus parámetros, lo que permite ejecutar la tarea más rápidamente, con menos requerimientos de hardware y a un coste menor. En la misma línea, la **cuantificación** permite reducir el tamaño del modelo, por ejemplo, de 16 bits a 4 bits, posibilitando que estos LLM puedan ser ejecutados en equipos individuales con GPUs más modestas, ampliando así su accesibilidad.

5. Posibles avances o líneas futuras

Este Trabajo de Fin de Grado ha permitido el desarrollo de un primer modelo basado en un LLM (Large Language Model) capaz de detectar noticias falsas a partir del análisis textual. No obstante, el ámbito de la inteligencia artificial evoluciona de manera constante y acelerada, lo que abre la puerta a múltiples líneas de mejora y futuras ampliaciones del sistema propuesto.

En este sentido, una de las principales limitaciones del modelo actual radica en su carácter exclusivamente textual. En la actualidad, una gran parte de la información, especialmente aquella procedente de redes sociales, se presenta acompañada de contenido multimedia como imágenes y vídeos. Estos elementos pueden ser utilizados para difundir información engañosa, incluso cuando el texto asociado resulta veraz transmitiendo asi malinformación. Asimismo, existen plataformas, como TikTok, en las que la información se transmite principalmente a través de contenido audiovisual, lo que resulta especialmente relevante dado su amplio uso entre la población joven.

Para abordar esta problemática, investigaciones recientes han propuesto modelos multimodales capaces de procesar simultáneamente distintos tipos de datos. Un ejemplo de ello es FND-LLM, que permite analizar no solo texto, sino también imágenes y vídeos, mejorando así la capacidad de detección de desinformación en entornos reales (Jingyuan, Zegju, Tianyi y Peiyang, 2025).

Por otro lado, existen enfoques que incorporan fuentes externas de conocimiento para verificar la veracidad de la información. Modelos como Milk-FD comparan el contenido analizado con datos procedentes de enciclopedias y recursos disponibles en internet, lo que contribuye a una validación más robusta de las afirmaciones presentes en las noticias.

Otra línea prometedora es la representada por modelos con capacidades de aprendizaje eficiente a partir de pocos ejemplos. En este contexto, DAFND introduce mecanismos de meta-aprendizaje que permiten al sistema adaptarse rápidamente a nuevos escenarios informativos sin necesidad de grandes volúmenes de datos de entrenamiento. Esto resulta especialmente útil en situaciones emergentes, como conflictos o eventos inesperados, donde la disponibilidad de datos etiquetados es limitada.

Finalmente, como posible línea de avance futuro, más allá de los progresos en los modelos de lenguaje de gran tamaño (LLM), podría plantearse la integración de estos en el

desarrollo de un modelo multimodal de gran capacidad, que podría ser implementado por plataformas digitales o compañías tecnológicas como Google.

De este modo, bajo cada publicación podría incorporarse una evaluación generada por el modelo que indique si el contenido es potencialmente verídico o no, tras analizar la información disponible. Es decir, un sistema que permita analizar directamente los contenidos publicados y ofrecer una estimación sobre su grado de veracidad, contribuyendo así a la detección de posibles noticias falsas.

Finalmente, otro posible avance futuro muy ambicioso y bonito, más allá de los avances en los LLM seria poder desarrollar un modelo de LLM muy potente con un entrenamiento actualizado constantemente, capaz de cruzar información, ver fotos, videos, etc y entrenado gracias a todos estos avances tecnológicos en los LLM que fuese introducido por las redes sociales o por compañías como Google o safari y que, debajo de cada contenido, se establezca que dice este LLM sobre si el contenido es falso o no. Es decir, que este modelo, analice directamente toda información publicada y establezca su veredicto sobre si la clasificaría como verdadera o no.

Por ejemplo, en la figura 45, se ilustra cómo podría funcionar un modelo diseñado para evaluar la veracidad de las noticias y su posible aplicación a gran escala en Internet. Se ha introducido en el modelo Falcon 3-1B previamente entrenado una noticia supuestamente publicada por "El País", con el titular "Nueva York es la capital de Estados Unidos" y el cuerpo "Nueva York es la capital de uno de los países del continente de América". En este caso, el modelo la ha clasificado como veraz. En cambio, al introducir una noticia atribuida a Laguna que afirma que "El Papa ha decidido que, en honor a Donald Trump y a sus intentos de establecer la paz, se va a cambiar el nombre de Papa León XIV a Papa Trump", el modelo la ha identificado como falsa.

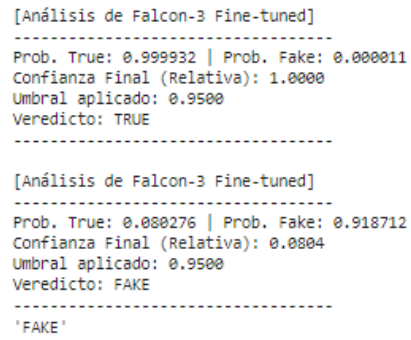


Figura 45: Ejemplo modelo detección de noticias falsas

Fuente: Elaboración propia

En conclusión, aunque el modelo desarrollado en este trabajo constituye un primer paso relevante en la detección automática de noticias falsas, la integración de capacidades multimodales, el uso de fuentes externas de verificación y la incorporación de técnicas de aprendizaje eficiente representan líneas clave para futuras mejoras y desarrollos.

6. Conclusiones

La investigación desarrollada en **este Trabajo de Fin de Grado permite confirmar que los Modelos de Lenguaje de Gran Escala (LLM) representan, hoy en día, uno de los acercamientos más sólidos y prometedores para mitigar el impacto de la desinformación**. A lo largo de este estudio, se ha evidenciado cómo la evolución de las arquitecturas de procesamiento de lenguaje ha pasado de un análisis puramente estadístico a una comprensión contextual que resulta clave para identificar las narrativas falsas que distorsionan la realidad social. Este resultado respalda la línea de investigación propuesta por autores como Jingyuan et al. (2025), quienes defienden que la detección efectiva debe evolucionar desde enfoques basados en la extracción manual de características hacia metodologías más robustas fundamentadas en modelos automáticos de aprendizaje.

Un pilar fundamental de este trabajo ha sido el Análisis Exploratorio de Datos (EDA), el cual permitió identificar patrones intrínsecos en la estructura de la desinformación. Gracias a este análisis previo, **se pudo constatar que las noticias falsas no solo difieren en contenido, sino también en forma**: tienden a presentar una mayor carga emocional, una extensión más reducida, un uso más frecuente de recursos lingüísticos de impacto en comparación con las noticias verdaderas e incluso datos faltantes en la noticia. Comprender estas variables fue determinante para orientar el entrenamiento de los modelos y entender por qué ciertas arquitecturas logran discernir la verdad con mayor eficacia.

Estos hallazgos son consistentes con los resultados de Horne y Adalı (2017), quienes señalan que la desinformación tiende a caracterizarse por un estilo lingüístico más simple y con mayor carga emocional en comparación con las noticias verídicas, que presentan estructuras más complejas.

No obstante, en este trabajo se ha optado por no emplear técnicas basadas exclusivamente en metadatos superficiales, como la fuente o ratios de puntuación. Esta decisión responde a la limitada robustez de estos enfoques frente a posibles ataques adversarios, ya que un generador de desinformación podría eludir la detección mediante la modificación de aspectos formales. En consecuencia, se prioriza el uso de modelos basados en la comprensión semántica profunda, como los LLM, que permiten capturar representaciones más ricas del contenido textual junto con variables tan determinantes como la fuente.

El proceso de experimentación llevado a cabo pone de manifiesto que **las mejoras tecnológicas contribuyen de forma directa a una clasificación más precisa y robusta**, lo cual se evidencia en el desempeño del modelo Falcon 3-1B, que alcanza una accuracy del 90%. No obstante, este trabajo introduce un matiz ético y práctico relevante: **el ajuste del umbral de clasificación al 95%.** Frente a la tendencia habitual en la literatura técnica, como la propuesta por Wang (2017), en la que se priorizan métricas globales como la exactitud o el F1-score, en esta investigación se plantea que el coste asociado a no detectar una noticia falsa es superior al del error contrario. En consecuencia, al configurar el modelo para clasificar como "verdadero" únicamente aquellos casos con un nivel de confianza elevado, se obtiene un **recall del 95% para la clase "fake",** priorizando así la capacidad de detección frente a la infodemia.

Si bien el modelo entrenado en este proyecto alcanzó un nivel de accuracy elevado, es fundamental reconocer que **estos sistemas aún no son perfectos**. Este margen de error residual nos recuerda que la detección de noticias falsas es un desafío dinámico; sin embargo, los resultados obtenidos confirman que cada avance en la optimización de los modelos nos sitúa un paso más cerca de una solución integral.

En conclusión, la transición hacia modelos como Falcon3-1B no solo supone un salto técnico en la capacidad de computación, sino una herramienta de valor incalculable para disminuir los problemas sociales y reputacionales derivados de la "infodemia". La evolución constante de estos modelos, sumada a los esfuerzos de investigación en el campo de la Inteligencia Artificial, ratifica que estamos en la senda correcta para construir un ecosistema digital más veraz y seguro. Aunque la perfección técnica sea un horizonte en movimiento, la trayectoria aquí analizada confirma que la tecnología ya es un aliado indispensable en la protección de la integridad informativa.

7. Declaración de Uso de Herramientas de Inteligencia Artificial Generativa en Trabajos Fin de Grado

ADVERTENCIA: Desde la Universidad consideramos que ChatGPT u otras herramientas similares son herramientas muy útiles en la vida académica, aunque su uso queda siempre bajo la responsabilidad del alumno, puesto que las respuestas que proporciona pueden no ser veraces. En este sentido, NO está permitido su uso en la elaboración del Trabajo fin de Grado para generar código porque estas herramientas no son fiables en esa tarea. Aunque el código funcione, no hay garantías de que metodológicamente sea correcto, y es altamente probable que no lo sea.

Por la presente, yo, Clara Oquiñena Goyena, estudiante de E2+Analytics de la Universidad Pontificia Comillas al presentar mi Trabajo Fin de Grado titulado "LLM para la detección de noticias falsas: La evolución de los modelos", declaro que he utilizado la herramienta de Inteligencia Artificial Generativa ChatGPT u otras similares de IAG de código sólo en el contexto de las actividades descritas a continuación [el alumno debe mantener solo aquellas en las que se ha usado ChatGPT o similares y borrar el resto. Si no se ha usado ninguna, borrar todas y escribir “no he usado ninguna”]:

1. **Brainstorming de ideas de investigación:** Utilizado para idear y esbozar posibles áreas de investigación.
2. **Crítico:** Para encontrar contra-argumentos a una tesis específica que pretendo defender.
3. **Referencias:** Usado conjuntamente con otras herramientas, como Science, para identificar referencias preliminares que luego he contrastado y validado.
4. **Metodólogo:** Para descubrir métodos aplicables a problemas específicos de investigación.
5. **Interpretador de código:** Para realizar análisis de datos preliminares.
6. **Estudios multidisciplinares:** Para comprender perspectivas de otras comunidades sobre temas de naturaleza multidisciplinar.
7. **Constructor de plantillas:** Para diseñar formatos específicos para secciones del trabajo.

8. **Corrector de estilo literario y de lenguaje:** Para mejorar la calidad lingüística y estilística del texto.
9. **Generador previo de diagramas de flujo y contenido:** Para esbozar diagramas iniciales.
10. **Sintetizador y divulgador de libros complicados:** Para resumir y comprender literatura compleja.
11. **Generador de problemas de ejemplo:** Para ilustrar conceptos y técnicas.
12. **Revisor:** Para recibir sugerencias sobre cómo mejorar y perfeccionar el trabajo con diferentes niveles de exigencia.
13. **Traductor:** Para traducir textos de un lenguaje a otro.

Afirmo que toda la información y contenido presentados en este trabajo son producto de mi investigación y esfuerzo individual, excepto donde se ha indicado lo contrario y se han dado los créditos correspondientes (he incluido las referencias adecuadas en el TFG y he explicitado para que se ha usado ChatGPT u otras herramientas similares). Soy consciente de las implicaciones académicas y éticas de presentar un trabajo no original y acepto las consecuencias de cualquier violación a esta declaración.

Fecha: 20/04/2026

Firma: _Clara Oquiñena Goyena_

8. Bibliografia

Abdullah, D.M., & Abdulazeez, A.M. (2021). Machine Learning Applications based on SVM Classification: A Review. *Qubahan Academic Journal*, 1(2). https://doi.org/10.48161/qaj.v1n2a50

Aguaded, I., & Romero-Rodríguez, L.M. (2015). Mediamorfosis y desinformación en la infoesfera: Alfabetización mediática, digital e informacional ante los cambios de hábitos de consumo informativo. *EKS, 16(1), 44-57*. http://dx.doi.org/10.14201/eks20151614457

Aragón, M. E., Jarquín, H., Gómez, M. M. Y., Escalante, H. J., Villaseñor-Pineda, L., Gómez-Adorno, H., ... & Posadas-Durán, J. P. (2020, September). Overview of mex-a3t at iberlef 2020: Fake news and aggressiveness analysis in mexican spanish. In Notebook Papers of 2nd SEPLN Workshop on Iberian Languages Evaluation Forum (IberLEF), Malaga, Spain.

Baines, D. & Elliot, R.J.R. (April 2020). Defining misinformation, disinformation and malinformation: An urgent need for clarity during the COVID-19 infodemic. *University of Birmingham.* Recuperado de /https://repec.cal.bham.ac.uk/pdf/20-06.pdf

Banco Mundial. (2025). *Individuos que utilizan internet (% de la población).* Recuperado de https://datos.bancomundial.org/indicador/IT.NET.USER.ZS

Bentéjac, C. , Csorgo, A. , Martínez-Muñoz, G. (2019). A comparative Analysis of XGBoost. *arXiv (Cornell Univeristy). doi:* https://doi.org/10.48550/arXiv.1911.01914

Bird, S. , Klein, E. , Loper, E. (2009). *Natural Language Processing with Python*. Sebastopol, United States: O'Reilly Media, Inc.

Couronné, R., Probst, P., & Boulesteix, A. (2018). Random forest versus logistic regression: a large-scale benchmark experiment. *BMC Bioinformatics*, *19*(1), 270. https://doi.org/10.1186/s12859-018-2264-5

Gómez-Adorno, H., Posadas-Durán, J. P., Enguix, G. B., & Capetillo, C. P. (2021). Overview of FakeDeS at IberLEF 2021: Fake News Detection in Spanish Shared Task. Procesamiento del Lenguaje Natural, 67, 223-231.

González Silot, S. y Martínez Cámara, E. , Luzón García, M.V. (2022).*fIlfA: Modelado computacional de la desinformación.* Universidad de Granada, España.

Horne, B. D., & Adali, S. (2017). This just in: Fake news packs a lot in title, uses simpler, repetitive content in text body, more similar to satire than real news. *Proceedings of the 2nd International Workshop on News and Public Opinion at ICWSM*. https://doi.org/10.48550/arXiv.1703.09398

Hu, B. , Mao, Z. & Zhang, Y. (2025). An overview of fake news detection: From a new perspective. *Fundamental Research, 5* (1), 332-346. doi: https://doi.org/10.1016/j.fmre.2024.01.017

INE – Instituto Nacional de Estadistica. (2025). *Hogares que tienen acceso a Internet y hogares que tienen ordenador. Porcentaje de menores usuarios de TIC.* Recuperado de https://www.ine.es/ss/Satellite?L=es_ES&c=INESeccion_C&cid=1259925529799&p=%5C&pagename=ProductosYServicios%2FPYSLayout&param1=PYSDetalle&param3=1259924822888

Jingyuan, Y. , Zeqiu, X. , Tianyi, H. & Peiyang, Y. (2025). Challenges and Innovations in LLM-Powered Fake News Detection: A Synthesis of Approaches and Future Directions, *GAIIS '25: Proceedings of the 2025 2nd International Conference on Generative Artificial Intelligence and Information Securit,* 87-93. doi: https://dl.acm.org/doi/full/10.1145/3728725.3728739

Kuiler, E. W. (2021). *Natural Language Processing (NLP)*. En L. A. Schintler & C. L. McNeely (Eds.), *Encyclopedia of Big Data* (pp. 1–3). Springer. doi: https://doi.org/10.1007/978-3-319-32001-4_250-1

Lin, Y., Zhang, J., Li, Y., Lin, G., & Liu, Y. (2024). *The rise of diffusion models in time-series forecasting*. arXiv. https://arxiv.org/pdf/2401.02191

Ma, X., Zhang, Y., Ding, K., Yang, J., Wu, J., & Fan, H. (2024). On fake news detection with LLM enhanced semantics mining. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 508–521. doi: https://doi.org/10.18653/v1/2024.emnlp-main.31

Mars, M. (2022). FromWord Embeddings to Pre-Trained Language Models: A State-of-the-Art Walkthrough. *Applied Science, 12(17). doi: https://doi.org/10.3390/app12178805*

Menéndez, J. (2025). Del monopolio de la información a la atomización de la difusión. *El observatorio, 3,* 99-106. Recuperado de https://revistas.unsta.edu.ar/index.php/OBS/article/view/1126/1398

Naveed, H. , Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., … Mian, A. (2024). A comprehensive overview of Large Language Models. *arXiv (Cornell Univeristy)* doi: https://doi.org/10.48550/arXiv.2307.06435

Newman, N. (2025). Resumen ejecutivo y hallazgos clave del informe de 2025. *Reuters Institute.* Recuperado de https://reutersinstitute.politics.ox.ac.uk/es/digital-news-report/2025/dnr-resumen-ejecutivo

Oficina de Ciencia y Tecnología del Congreso de los Diputados (Oficina C). (2023). *Informe C: Desinformación en la era digital*. https://oficinac.es/es/informes-c/desinformacion-era-digital

Posadas-Durán, J. P. (s.f.). *FakeNewsCorpusSpanish* [Conjunto de datos]. GitHub. https://github.com/jpposadas/FakeNewsCorpusSpanish

Posadas-Durán, J. P., Gómez-Adorno, H., Sidorov, G., & Escobar, J. J. M. (2019). Detection of fake news in a new corpus for the Spanish language. Journal of Intelligent & Fuzzy Systems, 36(5), 4869-4876.

Schaffer, C. (1993). Overfitting avoidance as bias. *Machine Learning, 10*, 153–178.

Star, R. S. (2025, 1 septiembre). Natural Language Processing (NLP) and its importance in AI. *Serenity Star*. Recuperado de https://serenitystar.ai/blog/natural-language-processing-nlp

Tătaru, G.-C., Domenteanu, A., Delcea, C., Florescu, M. S., Orzan, M., & Cotfas, L.-A. (2024). Navigating the Disinformation Maze: A Bibliometric Analysis of Scholarly Efforts. Information, 15(12), 742. https://doi.org/10.3390/info15120742

Wardle, C., & Derakhshan, H. (2017). *Information disorder: Toward an interdisciplinary framework for research and policy making* (Council of Europe

Report DGI(2017)09). Council of Europe. https://firstdraftnews.org/wp-content/uploads/2017/11/PREMS-162317-GBR-2018-Report-désinformation-1.pdf

Yang Wang, W. (2017). “Liar, Liar Pants on Fire”: A new Benchmark Dataset for Fake News Detection. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, (2),* 422-426. doi: 10.18653/v1/P17-2067

8. ANEXOS

Importo y descargo todas las librerias necesarias

```
!pip install pysentimiento
```

Mostrar salida oculta

```
!pip install -q transformers datasets peft accelerate bitsandbytes
```

```
!pip install spacy
!python -m spacy download es_core_news_sm
```

Mostrar salida oculta

```
pip install nltk scikit-learn textblob pysentimiento spacy seaborn
```

Mostrar salida oculta

```
!pip install -q -U transformers datasets peft accelerate bitsandbytes
```

```
!pip install -U transformers accelerate peft bitsandbytes datasets
```

Mostrar salida oculta

```
#Importamos todas las librerias necesarias para realizar el codigo
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import SnowballStemmer
nltk.download('punkt')
import matplotlib
from sklearn.feature_extraction.text import TfidfVectorizer
from textblob import TextBlob
from pysentimiento import create_analyzer
import spacy
from sklearn.feature_extraction.text import CountVectorizer
import re
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import torch
from datasets import Dataset
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay, precision_score, accuracy_sco
from transformers import (
    AutoTokenizer,
    AutoModelForCausalLM,
    TrainingArguments,
    Trainer,
    DataCollatorForLanguageModeling,
    BitsAndBytesConfig
)
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
import random
from sentence_transformers import SentenceTransformer
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif
import xgboost as xgb
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import os
from tqdm import tqdm
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
```

Mostrar salida oculta

Cargamos y preparamos los datos

Se cargan los tres ficheros Excel del dataset (train, test y development) y se comprueba que se han importado correctamente. Posteriormente, se combinan los conjuntos, ya que no se utilizará el de development, y se realiza una nueva partición del dataset en un 80% para entrenamiento y un 20% para test. Esto se hace con el objetivo de tener el maximo numero de observaciones tanto para entrenar como para examinar los modelos.

```
#Leemos los 3 excels
train=pd.read_excel("train.xlsx")
test=pd.read_excel("test.xlsx")
development=pd.read_excel("development.xlsx")
```

```
#Analizamos train
print(train.head(5))
train=train.drop("Id",axis=1)
train.describe()
```

```
   Id Category      Topic          Source  \
0   1     Fake  Education  El Ruinaversal
1   2     Fake  Education     Hay noticia
2   3     Fake  Education  El Ruinaversal
3   4     True  Education    EL UNIVERSAL
4   5     Fake  Education          Lamula

                                        Headline  \
0  RAE INCLUIRÁ LA PALABRA "LADY" EN EL DICCIONAR...
1           La palabra "haiga", aceptada por la RAE
2  YORDI ROSADO ESCRIBIRÁ Y DISEÑARÁ LOS NUEVOS L...
3  UNAM capacitará a maestros para aprobar prueba...
4  pretenden aprobar libros escolares con conteni...

                                            Text  \
0  RAE INCLUIRÁ LA PALABRA "LADY" EN EL DICCIONAR...
1  La palabra "haiga", aceptada por la RAE La Rea...
2  YORDI ROSADO ESCRIBIRÁ Y DISEÑARÁ LOS NUEVOS L...
3  UNAM capacitará a maestros para aprobar prueba...
4  Alerta: pretenden aprobar libros escolares con...

                                            Link
0  http://www.elruinaversal.com/2017/06/10/rae-in...
1  https://haynoticia.es/la-palabra-haiga-aceptad...
2  http://www.elruinaversal.com/2018/05/06/yordi-...
3  http://www.eluniversal.com.mx/articulo/nacion/...
4  https://redaccion.lamula.pe/2018/06/19/memoria...
```

	Category	Topic	Source	Headline	Text	Link
count	676	676	676	676	676	676
unique	2	9	134	676	676	676
top	Fake	Politics	El Dizque	MUERTE DE PETER EL ANGUILA POR SOBREDOSIS	MUERTE DE PETER EL ANGUILA POR SOBREDOSIS\nPet...	http://lavozpopular.com/muerte-peter-el-anguil...

```
#Analizamos development
print(development.head(5))
development=development.drop("Id",axis=1)
development.describe()
```

```
   Id Category      Topic          Source  \
0   1     Fake  Education  El Ruinaversal
1   2     True  Education         Heraldo
2   3     True  Education             abc
3   4     True  Education         El país
4   5     Fake  Education  El Ruinaversal

                                        Headline  \
0  MAESTRA DE *NUMBER* AÑOS QUE TUVO RELACIONES C...
1  Oxford lanza sus propios exámenes de certifica...
2  La RAE estudia incluir «machirulo» en el Dicci...
3   Malala Yousafzai anuncia que estudiará en Oxford
4  Nombran a Ricardo Arjona nuevo miembro de la R...

                                            Text  \
0  MAESTRA DE *NUMBER* AÑOS QUE TUVO RELACIONES C...
1  Oxford lanza sus propios exámenes de certifica...
2  La RAE estudia incluir «machirulo» en el Dicci...
3  Malala Yousafzai anuncia que estudiará en Oxfo...
4  Nombran a Ricardo Arjona nuevo miembro de la R...

                                            Link
0  http://www.elruinaversal.com/2017/06/04/maestr...
1  https://www.heraldo.es/noticias/sociedad/2017/...
2  https://www.abc.es/cultura/abci-estudia-inclui...
3  https://elpais.com/internacional/2017/08/17/ac...
4  http://www.elruinaversal.com/2017/06/25/nombra...
```

	Category	Topic	Source	Headline	Text	Link
count	295	295	295	295	295	295
unique	2	9	82	294	294	294
top	True	Politics	El Ruinaversal	Messi jugará con la Selección Española	Messi jugará con la Selección Española\nUna im...	https://haynoticia.es/messi-jugara-con-la-sele...
freq	153	97	39	2	2	2

```
#Analizamos test
print(test.head(5))
test=test.drop("Id",axis=1)
test.describe()
```

```
   Id Category     Topic         Source  \
0   1     True  Covid-19  El Economista
1   2     Fake  Política     El matinal
2   3     True  Política        El País
3   4     Fake  Política     AFPFactual
4   5     True  Sociedad   La Republica

                                        Headline  \
0                     Covid-19: mentiras que matan
1  El Gobierno podrá acceder a las IPs de los móv...
2  La comunidad musulmana catalana denuncia a Vox...
3                                            NaN
4  El censo poblacional 2018 tendrá un costo de $...

                                            Text  \
0  El control de la Covid-19 no es sólo un tema d...
1  El Gobierno de Pedro Sánchez y Pablo Iglesias ...
2  Las tres federaciones que agrupan al 90% de la...
3  Se han dado a conocer los datos electorales pr...
4  La primera fase del censo será virtual y solo ...

                                            Link
0  https://www.eleconomista.com.mx/opinion/Covid-...
1  https://www.elmatinal.com/espana-ultima-hora/e...
2  https://elpais.com/espana/elecciones-catalanas...
3                          https://perma.cc/GYE6-SPMB
4  https://www.larepublica.co/economia/el-censo-p...
```

	Category	Topic	Source	Headline	Text	Link
count	572	572	565	500	572	569
unique	2	7	205	500	572	569
top	True	Covid-19	AFPFactual	En 2014 una revista ya alertaba sobre el coron...	Fue en el mes de febrero de 2014 cuando la rev...	https://www.NoticiadelsoldelalagunA.com.mx/dob...

Se observa que en el conjunto de test existen algunas filas que no contienen los campos source o link, mientras que en los conjuntos de train y development todos los registros están completos. Por ello, se procede a eliminar las filas que presentan valores ausentes en estos campos.

```
#Miro como son las noticias con vacios
noticias_con_vacios = test[test.isnull().any(axis=1)]
print(noticias_con_vacios)
```

```
    Category     Topic      Source  \
3       Fake  Política  AFPFactual
5       Fake  Covid-19  AFPFactual
15      Fake  Sociedad  AFPFactual
16      True  Sociedad         NaN
31      Fake  Política  AFPFactual
..       ...       ...         ...
519     Fake  Covid-19  AFPFactual
520     Fake  Covid-19  AFPFactual
545     Fake  Política  AFPFactual
556     Fake  Covid-19  AFPFactual
569     Fake  Política  AFPFactual

                                            Headline  \
3                                                NaN
5                                                NaN
15                                               NaN
16   El escandaloso incremento del patrimonio de Ir...
31                                               NaN
..                                               ...
519                                              NaN
520                                              NaN
545                                              NaN
556                                              NaN
569                                              NaN

                                                 Text  \
3    Se han dado a conocer los datos electorales pr...
5    Boooomm\nMUJERES VACUNADAS DE COVID ESTÁN MOST...
15   COMUNICADO DEL VICEPRESIDENTE DE FEDERACIÓN AJ...
16   El todavía vicepresidente segundo del Gobierno...
31   Lean (y vean la imagen) con mucha atención:\n\...
..                                                ...
519  PERMITIRME UNA LICENCIA.\n\nEstimados Alfonso ...
520  COVID-19\n? Debido al posible colapso del Sist...
545  #EnExpress ?????????\n#CirculaEnRedes ??\n??Pr...
556  El coronavirus viajó por todo el mundo desde W...
569  Evidentemente, Barak Obama ha sido arrestado e...

                                                 Link
3                            https://perma.cc/GYE6-SPMB
5    https://www.facebook.com/901924190177223/posts...
15   https://www.facebook.com/groups/75737276142798...
16   https://www.libertaddigital.com/espana/2021-03...
31                           https://perma.cc/X857-49PP
..                                                  ...
519  https://www.facebook.com/photo.php?fbid=358188...
520  https://www.facebook.com/740410852763104/posts...
545                          https://perma.cc/9SV7-RYZK
556  Perma | Juan Rico - El coronavirus viajó por t...
569  Perma | Obama, Biden y la directora de la CIA,...

[75 rows x 6 columns]
```

```
#Miro si estas noticias con vacios son verdaderas o falsas
total_vacios = noticias_con_vacios.shape[0]
print("Total de noticias con vacíos:", total_vacios)
vacios_fake = noticias_con_vacios[noticias_con_vacios["Category"] == "Fake"].shape[0]
print("Noticias con vacíos y cateogry = Fake:", vacios_fake)
```

```
Total de noticias con vacíos: 75
Noticias con vacíos y cateogry = Fake: 73
```

```
#Convierto los vacios a Nan y los elimino
test= test.replace(r'^\s*$', np.nan, regex=True)
test= test.dropna()
print(test.describe())
```

```
       Category     Topic      Source  \
count       497       497         497
unique        2         7         204
top        True  Covid-19  AFPFactual
freq        284       190          32

                                                Headline  \
count                                                497
unique                                               497
top     En 2014 una revista ya alertaba sobre el coron...
freq                                                   1

                                                   Text  \
count                                               497
unique                                              497
```

```
top     Fue en el mes de febrero de 2014 cuando la rev...
freq                                                    1

                                                     Link
count                                                 497
unique                                                497
top     https://www.Noticiadelsoldelalaguna.com.mx/dob...
freq                                                    1
```

```
#Actualmente, la distribución de los datos es aproximadamente un 44% para train, un 20% para development y un 37% para test
#Combino los excels
full_df = pd.concat([train, development, test], ignore_index=True)
full_df['Category'] = full_df['Category'].astype(str)

#Dividir en train (80%) y test (20%)
df, test = train_test_split(
    full_df,
    test_size=0.2,
    random_state=42,
    shuffle=True,
    stratify=full_df['Category']
)

#Verificar tamaños
print(len(df), len(test))
```

```
1174 294
```

```
df.describe()
```

	Category	Topic	Source	Headline	Text	Link
count	1174	1174	1174	1174	1174	1174
unique	2	16	296	1172	1171	1173
top	True	Politics	El Dizque	Un audio robado hunde a Pablo Iglesias: cóctel...	Para todo sacan lo del populismo, ni siquiera ...	http://www.argumentopolitico.com/2016/07/grupo...
freq	620	263	115	2	2	2

```
test.describe()
```

	Category	Topic	Source	Headline	Text	Link
count	294	294	294	294	294	294
unique	2	15	114	293	294	293
top	True	Politics	El Dizque	La copa de amor de Peña Nieto y Angélica Rivera	Confirman a Cuauhtémoc Blanco como parte de la...	https://www.huffingtonpost.com.mx/2018/04/13/l...

```
#Se comprueba la distribución de las clases fake y true en los nuevos conjuntos generados, con el objetivo de asegurar que
test_category_data = test['Category']
conteo = pd.Series(test_category_data).value_counts()
print(conteo)

train_category_data = df['Category']
conteo = pd.Series(train_category_data).value_counts()
print(conteo)
```

```
Category
True    155
Fake    139
Name: count, dtype: int64
Category
True    620
Fake    554
Name: count, dtype: int64
```

En general, el número de ejemplos true y fake está bastante equilibrado en cada uno de los conjuntos de datos, lo cual es importante para el análisis posterior. Esto evita que el modelo se sesgue hacia la clase mayoritaria y tienda a clasificar por defecto como true debido a una distribución descompensada.

```
#Miramos si hay observaciones repetidas
repeated_df = df.duplicated().any()
print(repeated_df)
```

```
False
```

```
#Miramos si hay observaciones repetidas para el test
repeated_test = test.duplicated().any()
print(repeated_test)
```

```
False
```

```
#Reafirmamos que no hay vacios en ninguno de los dos conjuntos de datos
hay_vacios = df.isnull().any().any()
print(hay_vacios)
hay_vacios = test.isnull().any().any()
print(hay_vacios)
```

```
False
False
```

```
#Hago un duplicado de ambos datasets "limpios" por si acaso se necesita mas adelante
df_backup=df.copy()
test_backup=test.copy()
full_df_backup=full_df.copy()
```

Analisis exploratorio de los datos (EDA)

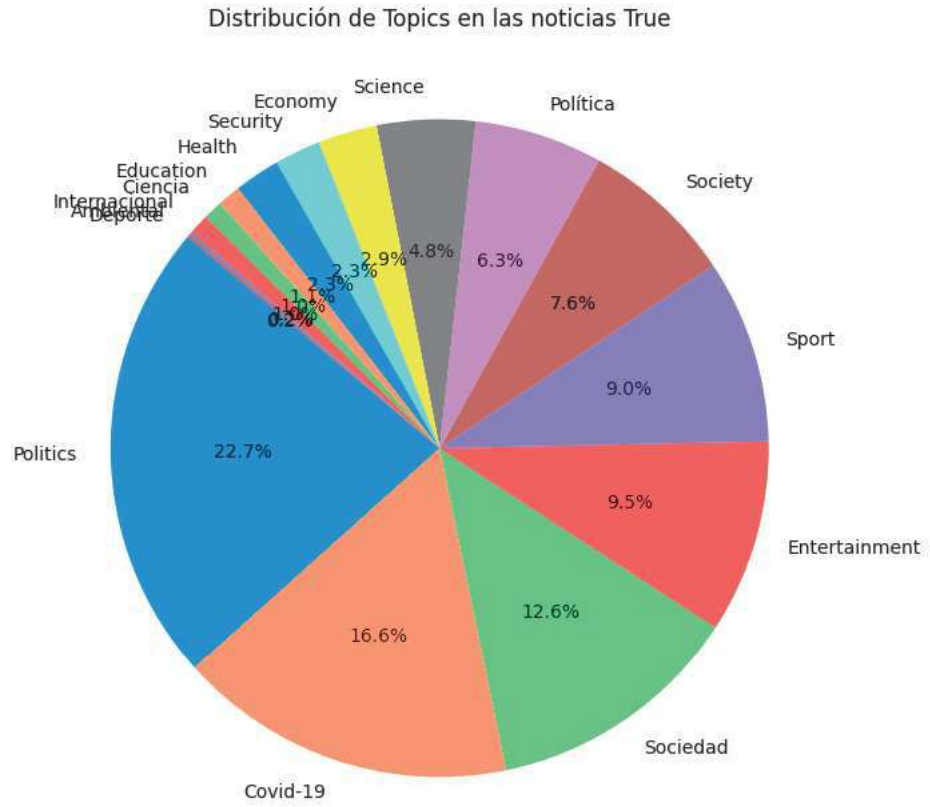
Para este apartado nos vamos a centrar en analizar patrones que pueda distinguir noticias verdaderas de noticias falsas. Para ello utilizare unicamente la base de datos de train (df)

```
#Separamos las noticias verdaderas de las falsas

# Filtramos las noticias verdaderas y hago copia
df_true = df[df['Category'] == 'True'].copy()
df_true_backup = df_true.copy()

# Filtramos las noticias falsas y hago copia
df_fake = df[df['Category'] == 'Fake'].copy()
df_fake_backup = df_fake.copy()
```

Analisis de frecuencias.

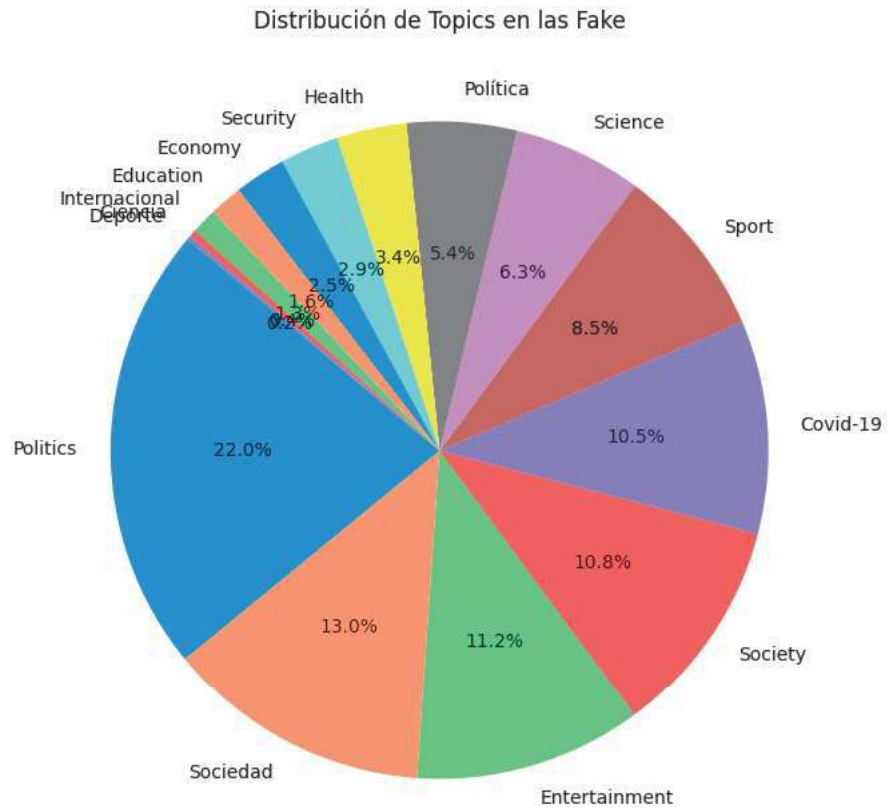
```
#Vemos cuantos hay de cada topic en las noticias true
frequency_topics = df_true['Topic'].value_counts()
print(frequency_topics)

#Lo grafico en un pie chart
plt.figure(figsize=(8,8))
plt.pie(frequency_topics, labels=frequency_topics.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribución de Topics en las noticias True')
plt.show()
```

```
Topic
Politics         141
Covid-19         103
Sociedad          78
Entertainment     59
Sport             56
Society           47
Política          39
Science           30
Economy           18
Security          14
Health            14
Education          7
Ciencia            6
Internacional      6
Ambiental          1
Deporte            1
Name: count, dtype: int64
```



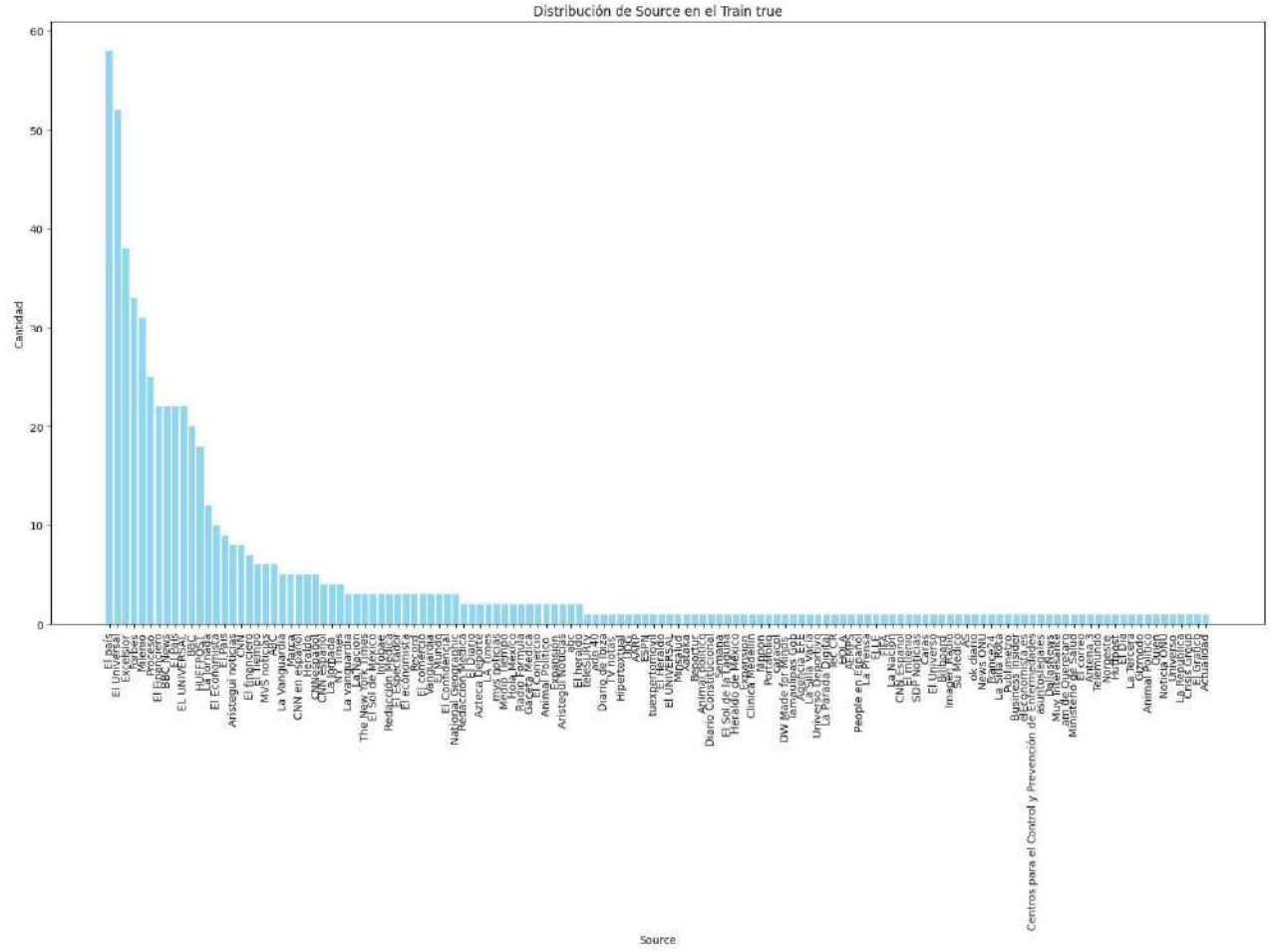

```
#Vemos cuantos hay de cada topic en las noticias fake
frequency_topics = df_fake['Topic'].value_counts()
print(frequency_topics)

#Lo grafico en un pie chart
plt.figure(figsize=(8,8))
plt.pie(frequency_topics, labels=frequency_topics.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribución de Topics en las Fake')
plt.show()
```

```
Topic
Politics         122
Sociedad          72
Entertainment     62
Society           60
Covid-19          58
Sport             47
Science           35
Política          30
Health            19
Security          16
Economy           14
Education          9
Internacional      7
Ciencia            2
Deporte            1
Name: count, dtype: int64
```



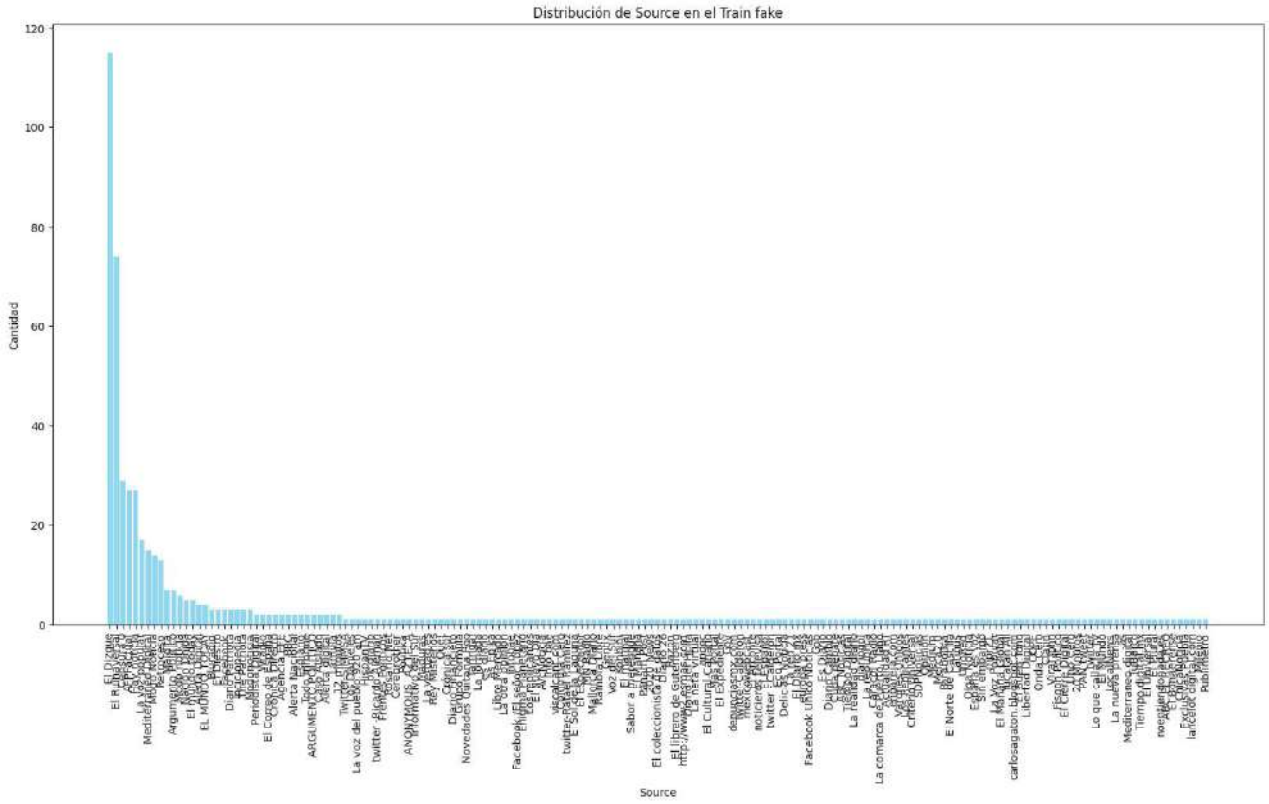

```
#Source true
frequency_source_true = df_true['Source'].value_counts()
print(frequency_source_true)

# Crear gráfico de barras
plt.figure(figsize=(20,10))
plt.bar(frequency_source_true.index, frequency_source_true.values, color='skyblue')
plt.xlabel('Source')
plt.ylabel('Cantidad')
plt.title('Distribución de Source en el Train true')
plt.xticks(rotation=90)
plt.show()
```

```
Source
El país          58
El Universal     52
Excelsior        38
Forbes           33
Milenio          31
                 ..
Universo          1
La Republica      1
Crisis Group      1
El Grafico        1
Actualidad        1
Name: count, Length: 134, dtype: int64
```

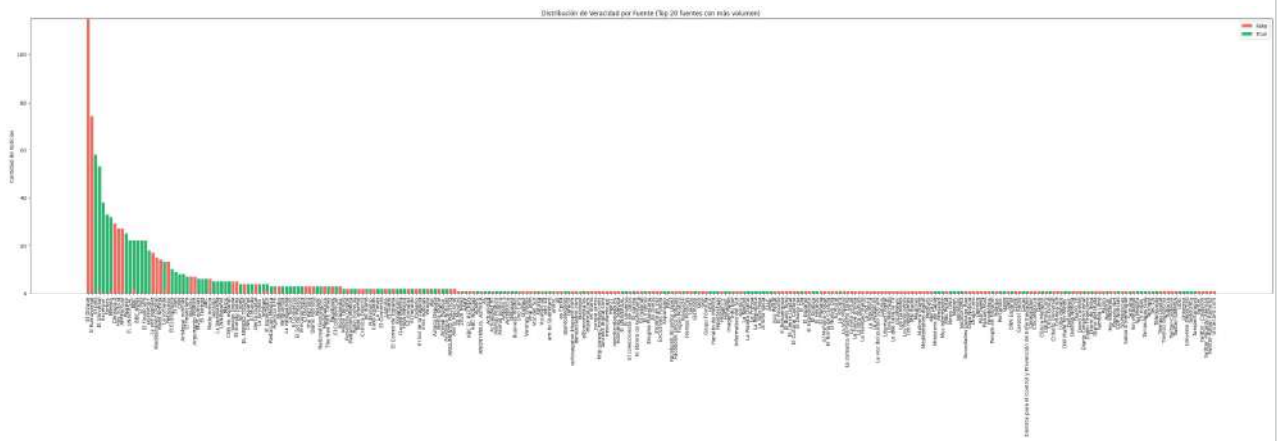


```
#Source fake
frequency_source_fake = df_fake['Source'].value_counts()
print(frequency_source_fake)

# Crear gráfico de barras
plt.figure(figsize=(20,10))
plt.bar(frequency_source_fake.index, frequency_source_fake.values, color='skyblue')
plt.xlabel('Source')
plt.ylabel('Cantidad')
plt.title('Distribución de Source en el Train fake')
plt.xticks(rotation=90)
plt.show()
```

```
Source
El Dizque                115
El Ruinaversal            74
Censura 0                 29
AFPFactual                27
Hay noticia               27
                        ...
ChicaNoticia               1
Exclusivas Puebla          1
lancelot digital.com       1
Milenio                    1
Publimetro                 1
Name: count, Length: 173, dtype: int64
```



```
#Junto los sources en un unico grafico de barras para ver si hay alguna source que se repita y que este tanto en noticias v

#Cargamos las variables
df_true_counts = frequency_source_true.reset_index()
df_true_counts.columns = ['Source', 'True_Count']

df_fake_counts = frequency_source_fake.reset_index()
df_fake_counts.columns = ['Source', 'Fake_Count']

#Unimos ambos conteos
df_fuentes = pd.merge(df_fake_counts, df_true_counts, on='Source', how='outer').fillna(0)
df_fuentes['Total'] = df_fuentes['Fake_Count'] + df_fuentes['True_Count']
df_plot = df_fuentes.sort_values(by='Total', ascending=False)

#Grafica
plt.figure(figsize=(35, 12))
plt.bar(df_plot['Source'], df_plot['Fake_Count'], color='red', label='Fake')
plt.bar(df_plot['Source'], df_plot['True_Count'], bottom=df_plot['Fake_Count'], color='green', label='True')
plt.xticks(rotation=90)
plt.ylabel('Cantidad de Noticias')
plt.title('Distribución de Veracidad por Fuente (Top 20 fuentes con más volumen)')
plt.legend()
plt.tight_layout()
plt.show()
```



Se observa que algunas fuentes presentan una combinación de noticias falsas y verdaderas; sin embargo, la mayoría de las fuentes tienden a publicar exclusivamente noticias verdaderas o exclusivamente noticias falsas. Este patrón puede constituir uno de los principales indicadores para la detección de la veracidad de una noticia.

DTM y TF-IDF

Ahora vamos a eliminar las stopwords del texto y de los headlines y ver cuales son las palabras mas frecuentes de tanto las noticias true como las noticias fake a traves del DTM

```
#Descargo los stopwords
nltk.download('stopwords')

#Las noticias estan en Español asi que las configuro para el español
stop_words = set(stopwords.words('spanish'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
#Ahora me voy a centrar en el analisis de las noticias como tal. Para ello voy a "limpiar" los textos y los titulares
#De las true
df_true["Text"] = df_true["Text"].str.lower()
df_true["Text"] = df_true["Text"].apply(lambda x: " ".join([word for word in x.split() if word not in stop_words]))
df_true["Headline"] = df_true["Headline"].str.lower()
df_true["Headline"] = df_true["Headline"].astype(str).fillna("")
df_true["Headline"] = df_true["Headline"].apply(lambda x: " ".join([word for word in x.split() if word not in stop_words]))
df_true["Text"] = df_true["Text"].str.replace(r"[^a-zA-Záéíóúüñ ]", "", regex=True)
df_true["Headline"] = df_true["Headline"].str.replace(r"[^a-zA-Záéíóúüñ ]", "", regex=True)

#Y de las false
df_fake["Text"] = df_fake["Text"].str.lower()
df_fake["Text"] = df_fake["Text"].apply(lambda x: " ".join([word for word in x.split() if word not in stop_words]))
df_fake["Headline"] = df_fake["Headline"].str.lower()
df_fake["Headline"] = df_fake["Headline"].astype(str).fillna("")
df_fake["Headline"] = df_fake["Headline"].apply(lambda x: " ".join([word for word in x.split() if word not in stop_words]))
df_fake["Text"] = df_fake["Text"].str.replace(r"[^a-zA-Záéíóúüñ ]", "", regex=True)
df_fake["Headline"] = df_fake["Headline"].str.replace(r"[^a-zA-Záéíóúüñ ]", "", regex=True)
```

```
#Creamos la DTM de los datos true
vectorizer = CountVectorizer()
DTM = vectorizer.fit_transform(df_true["Text"])
DTM_df = pd.DataFrame(DTM.toarray(), columns=vectorizer.get_feature_names_out())

frecuencias = DTM_df.sum(axis=0)
frecuencias_ordenadas = frecuencias.sort_values(ascending=False)
top20 = frecuencias_ordenadas.head(20)
print(top20)
```

```
number        2621
méxico         633
si             596
años           584
personas       503
ser            500
país           482
dijo           460
gobierno       435
dos            420
covid          382
parte          354
presidente     350
puede          338
año            335
después        332
según          329
así            327
millones       322
salud          308
dtype: int64
```

```
#Creamos la DTM de los datos false
vectorizer = CountVectorizer()
DTM = vectorizer.fit_transform(df_fake["Text"])
DTM_df = pd.DataFrame(DTM.toarray(), columns=vectorizer.get_feature_names_out())

frecuencias = DTM_df.sum(axis=0)
frecuencias_ordenadas = frecuencias.sort_values(ascending=False)
top20 = frecuencias_ordenadas.head(20)
print(top20)
```

```
number        876
si            514
méxico        396
ser           361
así           302
años          271
país          252
ahora         234
gobierno      230
además        228
personas      224
solo          221
presidente    216
según         207
```

```
dijo          205
puede         194
vez           186
parte         186
virus         180
hace          178
dtype: int64
```

Hago lo mismo pero con los headlines en vez de solo con el texto

```
#Creamos la DTM de los titulos de el conjunto de datos true
vectorizer = CountVectorizer()
DTM = vectorizer.fit_transform(df_true["Headline"])
DTM_df = pd.DataFrame(DTM.toarray(), columns=vectorizer.get_feature_names_out())

frecuencias = DTM_df.sum(axis=0)
frecuencias_ordenadas = frecuencias.sort_values(ascending=False)
top20 = frecuencias_ordenadas.head(20)
print(top20)
```

```
covid          64
number         44
méxico         34
coronavirus    34
amlo           32
trump          17
tras           15
vacuna         15
peña           15
cómo           14
vacunas        13
eu             12
años           12
mujeres        12
salinas        11
gobierno       11
nuevo          11
meade          11
primera        10
pide           10
dtype: int64
```

```
#Creamos la DTM de los titulos de el conjunto de datos fake
vectorizer = CountVectorizer()
DTM = vectorizer.fit_transform(df_fake["Headline"])
DTM_df = pd.DataFrame(DTM.toarray(), columns=vectorizer.get_feature_names_out())

frecuencias = DTM_df.sum(axis=0)
frecuencias_ordenadas = frecuencias.sort_values(ascending=False)
top20 = frecuencias_ordenadas.head(20)
print(top20)
```

```
number         54
amlo           43
méxico         34
gobierno       19
covid          19
años           19
coronavirus    18
peña           17
tras           15
meade          14
rivera         14
si             13
nueva          13
nieto          13
pri            13
ahora          12
millones       12
trump          12
américa        11
nuevo          11
dtype: int64
```

Se observa que las palabras más frecuentes son, en gran medida, similares en ambos casos, lo que sugiere que la matriz de términos-documento (DTM) no constituye una característica especialmente discriminativa para determinar si las noticias son verdaderas o falsas.

Se procederá a realizar el mismo análisis incorporando un proceso de lematización, con el objetivo de identificar los significados subyacentes de las palabras más frecuentes.

```
nlp = spacy.load("es_core_news_sm")
```

```
#Lematiazación headlines true
#Elimino estas palabras extras que salen mucho, no aportan valor y no estan en las stopwords
basura_extra = ["él", "number", "meadir", "peño", "año", "hacer", "ir"]

def lematizar_estricto(texto):
    if not isinstance(texto, str): return ""

    doc = nlp(texto.lower())
    lemas_limpios = []

    for token in doc:
        lema = token.lemma_.lower().strip()
        if (not token.is_stop and
            not token.is_punct and
            token.pos_ not in ["PRON", "DET", "ADP"] and
            not token.like_num and
            lema not in basura_extra and
            len(lema) > 2):
            lemas_limpios.append(lema)

    return " ".join(lemas_limpios)

#Procesamos el texto
df_true["Headline_Lemma"] = df_true["Headline"].apply(lematizar_estricto)
vectorizer = CountVectorizer(stop_words=basura_extra)

DTM = vectorizer.fit_transform(df_true["Headline_Lemma"])
DTM_df = pd.DataFrame(DTM.toarray(), columns=vectorizer.get_feature_names_out())

#Resultados finales
frecuencias = DTM_df.sum(axis=0).sort_values(ascending=False)
print("Top 20 Lemas headlines true (Limpieza Radical):")
print(frecuencias.head(20))
```

```
Top 20 Lemas headlines true (Limpieza Radical):
covid          64
méxico         34
amlo           32
coronavirus    28
trump          17
mujer          17
vacuna         14
pedir          14
elección       13
mexicano       11
peña           11
país           11
muerte         11
gobierno       11
nieto          10
político       10
millón         10
mundo           9
recibir         9
carmen          9
dtype: int64
```

```
#Lematiazación headlines fake
#Elimino estas palabras extras que salen mucho no aportan valor y no estan en las stopwords
basura_extra = ["él", "number", "meadir", "peño", "año", "hacer", "ir"]

def lematizar_estricto(texto):
    if not isinstance(texto, str): return ""

    doc = nlp(texto.lower())
    lemas_limpios = []

    for token in doc:
        lema = token.lemma_.lower().strip()
        if (not token.is_stop and
            not token.is_punct and
            token.pos_ not in ["PRON", "DET", "ADP"] and
            not token.like_num and
            lema not in basura_extra and
            len(lema) > 2):
            lemas_limpios.append(lema)

    return " ".join(lemas_limpios)

#Procesamos el texto
df_fake["Headline_Lemma"] = df_fake["Headline"].apply(lematizar_estricto)
vectorizer = CountVectorizer(stop_words=basura_extra)

DTM = vectorizer.fit_transform(df_fake["Headline_Lemma"])
DTM_df = pd.DataFrame(DTM.toarray(), columns=vectorizer.get_feature_names_out())

#Resultados finales
frecuencias = DTM_df.sum(axis=0).sort_values(ascending=False)
print("Top 20 Lemas headlines false (Limpieza Radical):")
print(frecuencias.head(20))
```

```
Top 20 Lemas headlines false (Limpieza Radical):
amlo           43
méxico         34
mexicano       22
mujer          21
covid          19
gobierno       19
coronavirus    14
nieto          14
pri            13
revelar        13
millón         12
pedir          12
morir          12
país           12
trump          12
facebook       11
español        11
recibir        11
permitir       10
nombre         10
dtype: int64
```

```
#Lematiazación texto true
#Elimino estas palabras extras que salen mucho no aportan valor y no estan en las stopwords
basura_extra = ["él", "number", "meadir", "peño", "año", "hacer", "ir"]

def lematizar_estricto(texto):
    if not isinstance(texto, str): return ""

    doc = nlp(texto.lower())
    lemas_limpios = []

    for token in doc:
        lema = token.lemma_.lower().strip()
        if (not token.is_stop and
            not token.is_punct and
            token.pos_ not in ["PRON", "DET", "ADP"] and
            not token.like_num and
            lema not in basura_extra and
            len(lema) > 2):
            lemas_limpios.append(lema)

    return " ".join(lemas_limpios)

#Procesamos el texto
df_true["Text_Lemma"] = df_true["Text"].apply(lematizar_estricto)
vectorizer = CountVectorizer(stop_words=basura_extra)

DTM = vectorizer.fit_transform(df_true["Text_Lemma"])
DTM_df = pd.DataFrame(DTM.toarray(), columns=vectorizer.get_feature_names_out())

#Resultados finales
frecuencias = DTM_df.sum(axis=0).sort_values(ascending=False)
print("Top 20 Lemas texto true (Limpieza Radical):")
print(frecuencias.head(20))
```

```
Top 20 Lemas texto true (Limpieza Radical):
país          674
méxico        633
persona       601
gobierno      469
caso          445
covid         381
mujer         373
público       371
presidente    350
mexicano      332
millón        321
salud         308
político      303
nacional      300
social        300
mundo         272
virus         272
ciudad        269
mes           264
partido       255
dtype: int64
```

```
#Lematiazación texto true
#Elimino estas palabras extras que salen mucho no aportan valor y no estan en las stopwords
basura_extra = ["él", "number", "meadir", "peño", "año", "hacer", "ir"]

def lematizar_estricto(texto):
    if not isinstance(texto, str): return ""

    doc = nlp(texto.lower())
    lemas_limpios = []

    for token in doc:
        lema = token.lemma_.lower().strip()
        if (not token.is_stop and
            not token.is_punct and
            token.pos_ not in ["PRON", "DET", "ADP"] and
            not token.like_num and
            lema not in basura_extra and
            len(lema) > 2):
            lemas_limpios.append(lema)

    return " ".join(lemas_limpios)

#Procesamos el texto
df_fake["Text_Lemma"] = df_fake["Text"].apply(lematizar_estricto)
vectorizer = CountVectorizer(stop_words=basura_extra)

DTM = vectorizer.fit_transform(df_fake["Text_Lemma"])
DTM_df = pd.DataFrame(DTM.toarray(), columns=vectorizer.get_feature_names_out())

#Resultados finales
frecuencias = DTM_df.sum(axis=0).sort_values(ascending=False)
print("Top 20 Lemas texto false (Limpieza Radical):")
print(frecuencias.head(20))
```

```
Top 20 Lemas texto false (Limpieza Radical):
méxico        396
país          342
persona       304
gobierno      255
mexicano      243
mujer         237
presidente    216
caso          197
dar           183
virus         179
mundo         175
gente         172
llegar        169
pasar         168
mes           161
partido       159
tiempo        158
nacional      151
ver           149
medio         145
dtype: int64
```

El DTM nos dice que palabras son las mas frecuentes en el corpus y el DTM con lematización nos dice que significados son mas frecuentes. Ahora, con el TF-IDF vamos a ver que palabras son las mas distintivas o importantes del corpus. Dicho TF-IDF se hara sobre las columnas lematizadas para captar los significados en lugar de las palabras ya que estas pueden difererir dependiendo del estilo de escritura.

```
#Lo realizo sobre los textos true
#Aseguramos que la columna lematizada no tenga nulos
df_true['Text_Lemma'] = df_true['Text_Lemma'].fillna('')

#Vectorizador TF-IDF
tfidf_vect = TfidfVectorizer(
    max_features=2000,
    ngram_range=(1, 2),
    sublinear_tf=True,
    stop_words=basura_extra
)

#Ajustamos y transformamos los datos lematizados
X_tfidf = tfidf_vect.fit_transform(df_true['Text_Lemma'])

#Extraemos la importancia promedio de cada término. Esto nos dirá qué significados definen mejor el corpus
weights = X_tfidf.mean(axis=0).tolist()[0]
features = tfidf_vect.get_feature_names_out()

#Creamos un DataFrame para ver el ranking
ranking_tfidf = pd.DataFrame({'término': features, 'importancia': weights})
ranking_tfidf = ranking_tfidf.sort_values(by='importancia', ascending=False)

#Visualización de resultados
print(f"Forma de la matriz TF-IDF: {X_tfidf.shape}")
print("\nTop 20 Significados más distintivos en texto true:")
print(ranking_tfidf.head(20))
```

```
Forma de la matriz TF-IDF: (620, 2000)

Top 20 Significados más distintivos en texto true:
         término  importancia
1250      méxico     0.027558
1379        país     0.025354
1403     persona     0.022998
895     gobierno     0.020634
1483  presidente     0.019976
265         caso     0.019715
1196    mexicano     0.019665
1778      social     0.018316
1566     público     0.018043
1257    nacional     0.017759
445        covid     0.017431
299       ciudad     0.016479
1238       mujer     0.016453
1205      millón     0.016441
1371     partido     0.016187
1193         mes     0.016050
1448    político     0.015913
1706       salud     0.015900
1242       mundo     0.015635
1605         red     0.015450
```

```
#Lo realizo sobre los textos fake
#Aseguramos que la columna lematizada no tenga nulos
df_fake['Text_Lemma'] = df_fake['Text_Lemma'].fillna('')

#Vectorizador TF-IDF
tfidf_vect = TfidfVectorizer(
    max_features=2000,
    ngram_range=(1, 2),
    sublinear_tf=True,
    stop_words=basura_extra
)

#Ajustamos y transformamos los datos lematizados
X_tfidf = tfidf_vect.fit_transform(df_fake['Text_Lemma'])

#Extraemos la importancia promedio de cada término. Esto nos dirá qué significados definen mejor al corpus
weights = X_tfidf.mean(axis=0).tolist()[0]
features = tfidf_vect.get_feature_names_out()

#Creamos un DataFrame para ver el ranking
ranking_tfidf = pd.DataFrame({'término': features, 'importancia': weights})
ranking_tfidf = ranking_tfidf.sort_values(by='importancia', ascending=False)

#Visualización de resultados
print(f"Forma de la matriz TF-IDF: {X_tfidf.shape}")
print("\nTop 20 Significados más distintivos en texto fake:")
print(ranking_tfidf.head(20))
```

```
Forma de la matriz TF-IDF: (554, 2000)

Top 20 Significados más distintivos en texto fake:
```

```
         término  importancia
1257      méxico     0.027047
1393        país     0.023849
1424     persona     0.020602
1197    mexicano     0.019649
869     gobierno     0.018700
1504  presidente     0.018573
1242       mujer     0.017443
264         caso     0.016624
489          dar     0.015973
1246       mundo     0.015717
859        gente     0.015284
1087      llegar     0.015251
1387       pasar     0.015249
1856      tiempo     0.014929
1939         ver     0.014928
1383     partido     0.014713
1792      social     0.014431
1456       poder     0.014411
1263    nacional     0.014319
1309     obrador     0.014230
```

```
#Lo realizo sobre los headlines fake
#Aseguramos que la columna lematizada no tenga nulos
df_fake['Headline_Lemma'] = df_fake['Headline_Lemma'].fillna('')

#Vectorizador TF-IDF
tfidf_vect = TfidfVectorizer(
    max_features=2000,
    ngram_range=(1, 2),
    sublinear_tf=True,
    stop_words=basura_extra
)

#Ajustamos y transformamos los datos lematizados
X_tfidf = tfidf_vect.fit_transform(df_fake['Headline_Lemma'])

#Extraemos la importancia promedio de cada término. Esto nos dirá qué significados definen mejor al corpus
weights = X_tfidf.mean(axis=0).tolist()[0]
features = tfidf_vect.get_feature_names_out()

#Creamos un DataFrame para ver el ranking
ranking_tfidf = pd.DataFrame({'término': features, 'importancia': weights})
ranking_tfidf = ranking_tfidf.sort_values(by='importancia', ascending=False)

#Visualización de resultados
print(f"Forma de la matriz TF-IDF: {X_tfidf.shape}")
print("\nTop 20 Significados más distintivos en headlineas fake:")
print(ranking_tfidf.head(20))
```

```
Forma de la matriz TF-IDF: (554, 2000)

Top 20 Significados más distintivos en headlineas fake:
         término  importancia
45          amlo     0.020446
1714      méxico     0.017464
1684    mexicano     0.011794
696        covid     0.011066
1704       mujer     0.010428
1546    gobierno     0.010228
674   coronavirus     0.009560
1699       morir     0.008279
1725       nieto     0.007144
1856     revelar     0.007132
1812         pri     0.007079
1768        país     0.006870
1840     recibir     0.006657
1770       pedir     0.006649
1559      guerra     0.006641
1778    permitir     0.006401
1304     español     0.006307
1429    facebook     0.006280
939    descubrir     0.005998
59      angélico     0.005992
```

```
#Lo realizo sobre los headlines true
#Aseguramos que la columna lematizada no tenga nulos
df_true['Headline_Lemma'] = df_true['Headline_Lemma'].fillna('')

#Vectorizador TF-IDF
tfidf_vect = TfidfVectorizer(
    max_features=2000,
    ngram_range=(1, 2),
    sublinear_tf=True,
    stop_words=basura_extra
)

#Ajustamos y transformamos los datos lematizados
X_tfidf = tfidf_vect.fit_transform(df_true['Headline_Lemma'])

#Extraemos la importancia promedio de cada término. Esto nos dirá qué significados definen mejor al corpus
weights = X_tfidf.mean(axis=0).tolist()[0]
features = tfidf_vect.get_feature_names_out()

#Creamos un DataFrame para ver el ranking
ranking_tfidf = pd.DataFrame({'término': features, 'importancia': weights})
ranking_tfidf = ranking_tfidf.sort_values(by='importancia', ascending=False)

#Visualización de resultados
print(f"Forma de la matriz TF-IDF: {X_tfidf.shape}")
print("\nTop 20 Significados más distintivos en headlines true:")
print(ranking_tfidf.head(20))
```

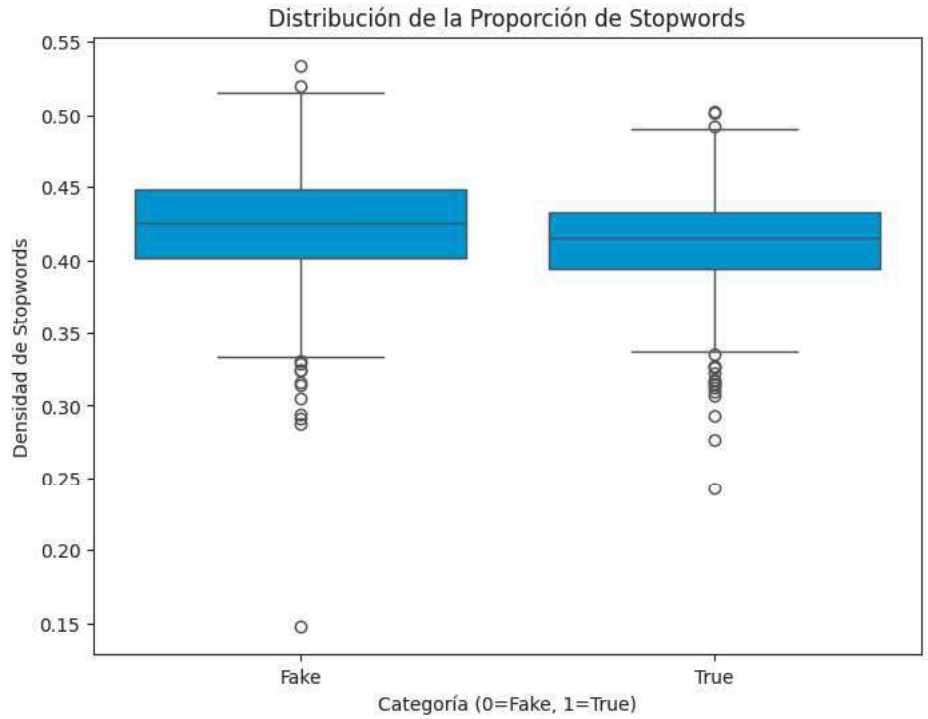
```
Forma de la matriz TF-IDF: (620, 2000)

Top 20 Significados más distintivos en headlines true:
          término  importancia
782         covid     0.023813
53           amlo     0.015462
1723       méxico     0.014986
747   coronavirus     0.012938
1925        trump     0.009161
1716        mujer     0.008706
1769        pedir     0.007556
1933       vacuna     0.007294
1719      mundial     0.007024
1715       muerte     0.006565
1700     mexicano     0.006308
1249     elección     0.006056
1767         país     0.006005
70        aprobar     0.005977
1609     gobierno     0.005812
1668       llegar     0.005784
1785         peña     0.005635
1624         hijo     0.005488
1713        morir     0.005438
367        carmen     0.005348
```

```
#Analizo cuantas stopwrods hay por noticia de media en las noticias true y en las noticias fake
# Descargamos las stopwords en español (o inglés, según tus noticias)
nltk.download('punkt')
nltk.download('punkt_tab')

def calculate_sw_proportion(Text):
    if not isinstance(Text, str) or len(Text) == 0:
        return 0
    words = nltk.word_tokenize(Text.lower())
    total_words = len(words)

    if total_words == 0:
        return 0

    sw_count = len([w for w in words if w in stop_words])
    return sw_count / total_words

df['sw_density'] = df['Text'].apply(calculate_sw_proportion)

proporcion_media = df.groupby('Category')['sw_density'].mean()

print("Proporción media de stopwords (0 a 1):")
print(proporcion_media)

#Para verlo en porcentaje
print("\nEn porcentaje:")
print(proporcion_media * 100)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
```

```
[nltk_data]   Package punkt_tab is already up-to-date!
Proporción media de stopwords (0 a 1):
Category
Fake    0.422317
True    0.412141
Name: sw_density, dtype: float64

En porcentaje:
Category
Fake    42.231701
True    41.214066
Name: sw_density, dtype: float64
```

```
#Lo grafico en un boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(x='Category', y='sw_density', data=df)
plt.title('Distribución de la Proporción de Stopwords')
plt.xlabel('Categoría (0=Fake, 1=True)')
plt.ylabel('Densidad de Stopwords')
plt.show()
```



Analisis de sentimientos y emociones

Para analizar los sentimientos es mejor realizarlo con los datos sin haber quitado stopwrods, etc. Por ello, voy a utilizar las copias que he hecho a df_true y df_fake antes de haber trabajado con ellas.

```
df_true=df_true_backup
df_fake=df_fake_backup
```

```
analyzer_pol = create_analyzer(task="sentiment", lang="es") #Pos, Neg, Neu
analyzer_emo = create_analyzer(task="emotion", lang="es") #6 emociones
```

```
Loading weights: 100%  201/201 [00:00<00:00, 616.76it/s]
Loading weights: 100%  201/201 [00:00<00:00, 5837.60it/s]
```

```
#Función para procesar los sentimientos y las emociones
def analizar_pysentimiento(texto):
    texto= str(texto).strip()
    if texto == "" or texto == "nan":
        return None

    #Predecimos polaridad y emociones
    pol = analyzer_pol.predict(texto)
    emo = analyzer_emo.predict(texto)

    #Calculo probabilidad de cada sentimiento y emocion
    res = {**pol.probas, **emo.probas}
    res['polaridad_final'] = pol.output
    res['emocion_final'] = emo.output
    return res
```

```
#Analizamos el titular de las noticias true
resultados = df_true['Headline'].apply(analizar_pysentimiento).apply(pd.Series)

#Unimos los resultados al dataframe original
df_final = pd.concat([df_true, resultados], axis=1)

#Creamos la lista de columnas que queremos
cols_interes = [
    'Headline', 'Category',
    'POS', 'NEG', 'NEU',
    'joy', 'sadness', 'fear', 'anger', 'surprise', 'disgust',
    'polaridad_final', 'emocion_final'
]

print(df_final[cols_interes].head())
```

```
                                          Headline Category       POS  \
191          Investigan robo a casa de Andrea Legarreta     True  0.139790
963             Critican libro de Yordi Rosado en redes     True  0.020543
1380                 Viaje a la Polonia de la homofobia     True  0.054239
1011              La política tóxica contamina a España     True  0.004974
1044  Pasaporte Covid: el Consejo de Europa avisa de...     True  0.027956

           NEG       NEU       joy   sadness      fear     anger  surprise  \
191   0.428008  0.432202  0.006869  0.005756  0.028841  0.041586  0.073421
963   0.709318  0.270139  0.000930  0.002073  0.001443  0.000934  0.006967
1380  0.625246  0.320515  0.033375  0.009413  0.003219  0.001267  0.012390
1011  0.968906  0.026120  0.022751  0.063221  0.013415  0.458635  0.001458
1044  0.530603  0.441440  0.000699  0.001022  0.004725  0.001225  0.004689

       disgust polaridad_final emocion_final
191   0.036277             NEU        others
963   0.001007             NEG        others
1380  0.002727             NEG        others
1011  0.268686             NEG         anger
1044  0.000874             NEG        others
```

```
#Seleccionamos solo las columnas numéricas de sentimientos y emociones
cols_numericas = ['POS', 'NEG', 'NEU', 'joy', 'sadness', 'fear', 'anger', 'surprise', 'disgust']

#Calculamos la media
nota_final = df_final[cols_numericas].mean() * 100

print("---NOTA FINAL DEL CORPUS HEADLINE TRUE(Promedio de Probabilidades) ---")
print(nota_final.sort_values(ascending=False))

#Ver cuál es la emoción predominante más frecuente
conteo_emociones = df_final['emocion_final'].value_counts(normalize=True) * 100
print("\n--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---")
print(conteo_emociones)
```

```
---NOTA FINAL DEL CORPUS HEADLINE TRUE(Promedio de Probabilidades) ---
NEU         43.569220
NEG         42.261246
POS         14.169534
joy          2.888240
anger        2.712343
sadness      2.125259
surprise     1.890562
fear         1.136445
disgust      0.973688
dtype: float64

--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---
emocion_final
others      93.225806
anger        2.419355
joy          2.096774
```

```
sadness      1.612903
fear         0.322581
surprise     0.161290
disgust      0.161290
Name: proportion, dtype: float64
```

La gran mayoria de las veces los titulares True tienen una emocion neutral y ademas un sentimiento neutral (others)

```
resultados = df_fake['Headline'].apply(analizar_pysentimiento).apply(pd.Series)

#Unimos los resultados al dataframe original
df_final = pd.concat([df_fake, resultados], axis=1)

#Creamos la lista de columnas que queremos
cols_interes = [
    'Headline', 'Category',
    'POS', 'NEG', 'NEU',
    'joy', 'sadness', 'fear', 'anger', 'surprise', 'disgust',
    'polaridad_final', 'emocion_final'
]

#Mostramos el head del DataFrame final con todas estas columnas
print(df_final[cols_interes].head())
```

```
                                          Headline Category       POS  \
1052  Vicepresidenta denuncia infiltración del narco...     Fake  0.081671
1288  BIL GATES DETENIDO POR TERRORISMO BIOLOGICO, L...     Fake  0.065504
42    Aunque ya fue aprobado, los americanistas rech...     Fake  0.023023
992   Colau permite que ‘Tsunami’ anuncie en las mar...     Fake  0.008071
1410  Coronavirus Noam Chomsky: USA necesitaba imper...     Fake  0.047676

           NEG       NEU       joy   sadness      fear     anger  surprise  \
1052  0.520058  0.398271  0.001347  0.001107  0.007479  0.004773  0.006942
1288  0.488036  0.446460  0.005876  0.003840  0.011225  0.015277  0.013122
42    0.735997  0.240979  0.001051  0.000897  0.000371  0.000443  0.000752
992   0.919569  0.072360  0.002088  0.004172  0.013669  0.009129  0.028595
1410  0.546288  0.406036  0.001109  0.001004  0.001464  0.001293  0.004171

       disgust polaridad_final emocion_final
1052  0.005010             NEG        others
1288  0.017155             NEG        others
42    0.000240             NEG        others
992   0.004059             NEG        others
1410  0.000484             NEG        others
```

```
#Seleccionamos solo las columnas numéricas de sentimientos y emociones
cols_numericas = ['POS', 'NEG', 'NEU', 'joy', 'sadness', 'fear', 'anger', 'surprise', 'disgust']

# Calculamos la media
nota_final = df_final[cols_numericas].mean() * 100

print("--- NOTA FINAL DEL CORPUS headline fake(Promedio de Probabilidades) ---")
print(nota_final.sort_values(ascending=False))

#Ver cuál es la emoción predominante más frecuente
conteo_emociones = df_final['emocion_final'].value_counts(normalize=True) * 100
print("\n--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---")
print(conteo_emociones)
```

```
--- NOTA FINAL DEL CORPUS headline fake(Promedio de Probabilidades) ---
NEG         47.974651
NEU         40.536226
POS         11.489122
anger        4.646933
surprise     2.163067
disgust      1.828525
sadness      1.765401
joy          1.720431
fear         1.249687
dtype: float64

--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---
emocion_final
others      92.238267
anger        4.512635
joy          1.083032
sadness      0.722022
surprise     0.722022
disgust      0.541516
fear         0.180505
Name: proportion, dtype: float64
```

```
#Función para procesar cada texto de las noticias true
resultados = df_true['Text'].apply(analizar_pysentimiento).apply(pd.Series)

# Unimos los resultados al dataframe original
df_final = pd.concat([df_true, resultados], axis=1)

#Creamos la lista de columnas que queremos
cols_interes = [
    'Text', 'Category',
    'POS', 'NEG', 'NEU',
    'joy', 'sadness', 'fear', 'anger', 'surprise', 'disgust',
    'polaridad_final', 'emocion_final'
]

#Mostramos el head del DataFrame final con todas estas columnas
print(df_final[cols_interes].head())
```

```
                                                  Text Category       POS  \
191   Investigan robo a casa de Andrea Legarreta\nJo...     True  0.042564
963   Critican libro de Yordi Rosado en redes\nUsuar...     True  0.009940
1380  Cezary Nieradko posa para la foto. Los puños c...     True  0.017108
1011  Ya no hay un solo día de tregua en la política...     True  0.006242
1044  El apoyo al ‘pasaporte Covid-19’ no es unánime...     True  0.013748

           NEG       NEU       joy   sadness      fear     anger  surprise  \
191   0.665772  0.291664  0.004682  0.036881  0.030136  0.571744  0.040972
963   0.802583  0.187477  0.001013  0.009818  0.003069  0.029973  0.003181
1380  0.796187  0.186705  0.003688  0.252809  0.001690  0.320953  0.001510
1011  0.930968  0.062790  0.001476  0.042635  0.001192  0.468813  0.001383
1044  0.835653  0.150599  0.002806  0.029255  0.002928  0.117752  0.004163

       disgust polaridad_final emocion_final
191   0.178877             NEG         anger
963   0.007828             NEG        others
1380  0.051601             NEG        others
1011  0.028562             NEG         anger
1044  0.011887             NEG        others
```

```
#Seleccionamos solo las columnas numéricas de sentimientos y emociones
cols_numericas = ['POS', 'NEG', 'NEU', 'joy', 'sadness', 'fear', 'anger', 'surprise', 'disgust']

#Calculamos la media
nota_final = df_final[cols_numericas].mean() * 100
print("--- NOTA FINAL DEL CORPUS TRUE (Promedio de Probabilidades) ---")
print(nota_final.sort_values(ascending=False))

#Ver cuál es la emoción predominante más frecuente
conteo_emociones = df_final['emocion_final'].value_counts(normalize=True) * 100
print("\n--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---")
print(conteo_emociones)
```

```
--- NOTA FINAL DEL CORPUS TRUE (Promedio de Probabilidades) ---
NEG         44.371923
NEU         40.006664
POS         15.621413
anger        9.933687
sadness      7.779720
joy          3.707440
disgust      1.525027
fear         0.725783
surprise     0.706295
dtype: float64

--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---
emocion_final
others     81.612903
anger       9.838710
sadness     5.322581
joy         3.064516
fear        0.161290
Name: proportion, dtype: float64
```

```
#Función para procesar cada texto de las noticias fake
resultados = df_fake['Text'].apply(analizar_pysentimiento).apply(pd.Series)

#Unimos los resultados al dataframe original
df_final = pd.concat([df_fake, resultados], axis=1)

#Creamos la lista de columnas que queremos
cols_interes = [
    'Text', 'Category',
    'POS', 'NEG', 'NEU',
    'joy', 'sadness', 'fear', 'anger', 'surprise', 'disgust',
    'polaridad_final', 'emocion_final'
]

#Mostramos el head del DataFrame final con todas estas columnas
print(df_final[cols_interes].head())
```

```
                                                   Text Category       POS  \
1052  Marta Lucía Ramírez advirtió que tiene conocim...     Fake  0.034699
1288  El multimillonario fundador de Microsoft, Bill...     Fake  0.095301
42    Aunque ya fue aprobado, los americanistas rech...     Fake  0.011071
992   La plataforma separatista ‘Tsunami democràtic’...     Fake  0.019060
1410  El activista, filósofo, politólogo y lingüista...     Fake  0.082106

           NEG       NEU       joy   sadness      fear     anger  surprise  \
1052  0.504512  0.460789  0.000357  0.011886  0.008723  0.005529  0.005283
1288  0.195096  0.709603  0.005547  0.010216  0.013408  0.041265  0.021580
42    0.943840  0.045089  0.026325  0.111362  0.028313  0.556647  0.070849
992   0.747389  0.233551  0.002313  0.021499  0.007101  0.303479  0.006288
1410  0.604735  0.313159  0.003995  0.054612  0.005026  0.051815  0.002414

       disgust polaridad_final emocion_final
1052  0.001850             NEG        others
1288  0.015558             NEU        others
42    0.056408             NEG         anger
992   0.026981             NEG        others
1410  0.007130             NEG        others
```

```
#Seleccionamos solo las columnas numéricas de sentimientos y emociones
cols_numericas = ['POS', 'NEG', 'NEU', 'joy', 'sadness', 'fear', 'anger', 'surprise', 'disgust']

#Calculamos la media
nota_final = df_final[cols_numericas].mean() * 100

print("--- NOTA FINAL DEL CORPUS FAKE (Promedio de Probabilidades) ---")
print(nota_final.sort_values(ascending=False))

#Ver cuál es la emoción predominante más frecuente
conteo_emociones = df_final['emocion_final'].value_counts(normalize=True) * 100
print("\n--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---")
print(conteo_emociones)
```

```
--- NOTA FINAL DEL CORPUS FAKE (Promedio de Probabilidades) ---
NEG         55.267566
NEU         32.650702
anger       19.859573
POS         12.081731
sadness      8.792720
disgust      2.705496
joy          2.084862
surprise     0.991096
fear         0.941633
dtype: float64

--- PORCENTAJE DE APARICIÓN COMO EMOCIÓN DOMINANTE ---
emocion_final
others      70.938628
anger       21.480144
sadness      5.595668
joy          1.444043
fear         0.361011
surprise     0.180505
Name: proportion, dtype: float64
```

```
#Ahora hago un analisis sin separar true y false para luego integrarlo para los modelos de clasificación
df=df_backup
df['Headline'] = df['Headline'].astype(str).fillna('')
df['Text'] = df['Text'].astype(str).fillna('')

#Analizamos tanto los headlines como los textos
resultados_headline = df['Headline'].apply(analizar_pysentimiento).apply(pd.Series).add_suffix('_h')
resultados_text = df['Text'].apply(analizar_pysentimiento).apply(pd.Series)
```

```
```

Ahora vamos a calcular metadatos

```
#Utilizo la copia por si acaso pero realmente no seria necesario porque ya lo he hecho justo arriba, antes de analizar los
df=df_backup
```

```
#Definimos la función para extraer los metadatos
def extraer_metadatos(df, columna_texto):
    df_meta = pd.DataFrame(index=df.index)

    df_meta['longitud_char'] = df[columna_texto].str.len()
    df_meta['n_palabras'] = df[columna_texto].apply(lambda x: len(str(x).split()))

    def ratio_mayus(texto):
        letras = re.findall(r'[a-zA-Z]', str(texto))
        if not letras: return 0
        mayus = sum(1 for c in letras if c.isupper())
        return mayus / len(letras)

    df_meta['ratio_mayusculas'] = df[columna_texto].apply(ratio_mayus)

    df_meta['n_exclamaciones'] = df[columna_texto].apply(lambda x: str(x).count('!'))
    df_meta['n_interrogaciones'] = df[columna_texto].apply(lambda x: str(x).count('?'))

    df_meta['n_numeros'] = df[columna_texto].apply(lambda x: len(re.findall(r'\d+', str(x))))

    df_meta['avg_word_length'] = df_meta['longitud_char'] / (df_meta['n_palabras'] + 1)

    return df_meta

#Aplicamos la función para todo el conjunto de datos
meta_train_text = extraer_metadatos(df, 'Text')

print("Metadatos extraidos de los textos:")
print(meta_train_text.head())
```

```
Metadatos extraidos de los textos:
      longitud_char  n_palabras  ratio_mayusculas  n_exclamaciones  \
1052            757         120          0.031667                0
191            2060         329          0.019631                0
1288           3538         558          0.031073                0
963            1672         281          0.032975                0
1380           8740        1459          0.032512                0

      n_interrogaciones  n_numeros  avg_word_length
1052                  0          1         6.256198
191                   0          0         6.242424
1288                  0         12         6.329159
963                   0          0         5.929078
1380                  1         15         5.986301
```

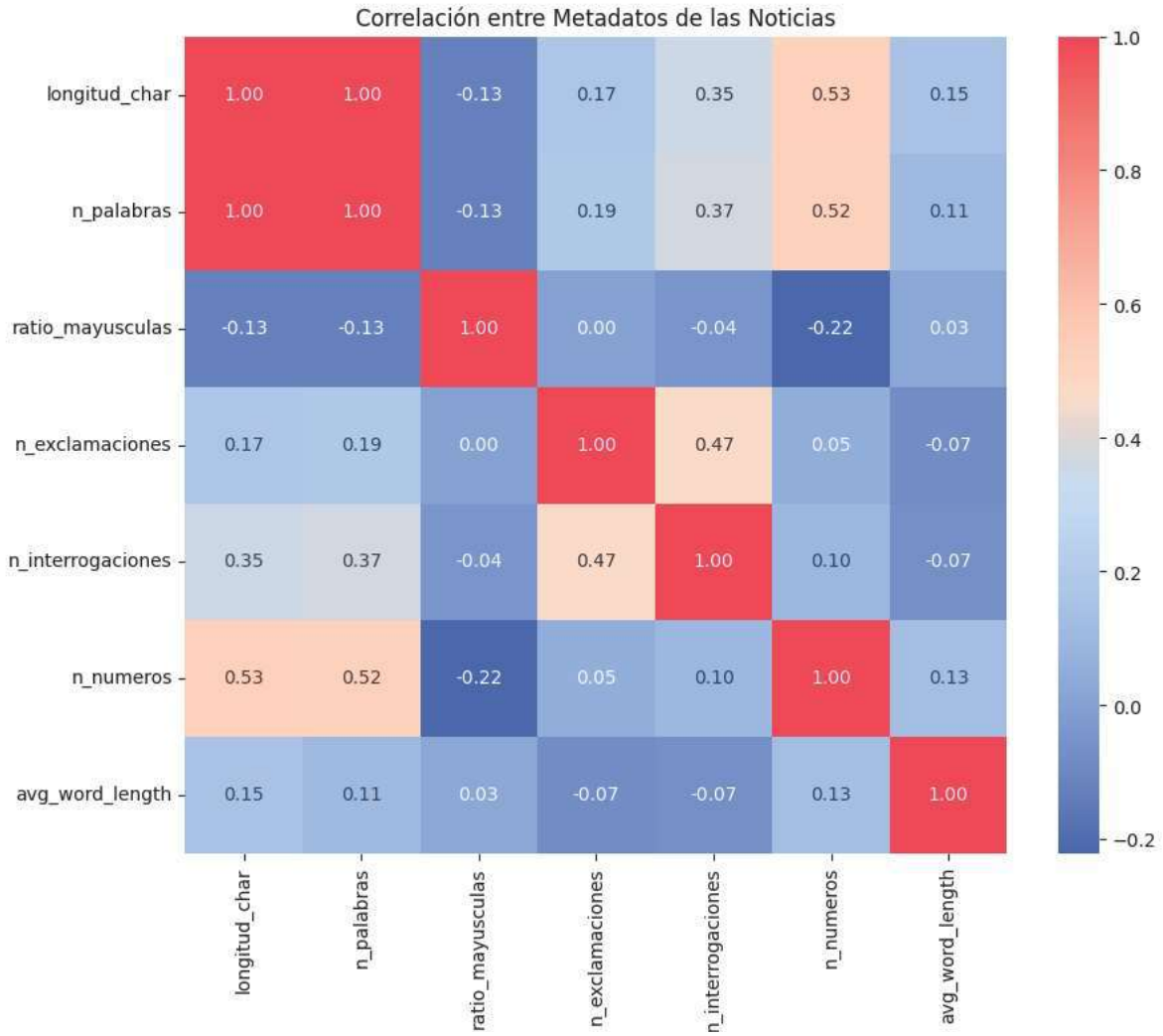
```
#Ahora analizamos las diferencias entras las noticias True y las noticias Fake
df_analisis = pd.concat([meta_train_text, df['Category']], axis=1)
metricas = ['ratio_mayusculas', 'n_exclamaciones', 'n_numeros', 'avg_word_length', 'n_interrogaciones', 'n_palabras', 'longi

#Grafico
fig, axes = plt.subplots(4, 2, figsize=(15, 12))
fig.suptitle('Comparativa de Metadatos: Noticias Verdaderas vs Falsas', fontsize=16)

for i, metrica in enumerate(metricas):
    row = i // 2
    col = i % 2
    sns.boxplot(x='Category', y=metrica, data=df_analisis, ax=axes[row, col], palette='Set2')
    axes[row, col].set_title(f'Distribución de: {metrica}')
    axes[row, col].set_xlabel('Categoría')
    axes[row, col].set_ylabel('Valor')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

Mostrar salida oculta

```
#Calculamos la media de todas las métricas
resumen_metadatos = df_analisis.groupby('Category').mean()

print("--- VALORES MEDIOS POR CATEGORÍA ---")
display(resumen_metadatos)
```

```
--- VALORES MEDIOS POR CATEGORÍA ---
           longitud_char  n_palabras  ratio_mayusculas  n_exclamaciones  n_interrogaciones  n_numeros  avg_word_length
Category
```

```
#Analizamos si hay correlaciones entre estos datos
plt.figure(figsize=(10, 8))
sns.heatmap(df_analisis.drop('Category', axis=1).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlación entre Metadatos de las Noticias')
plt.show()
```



Ahora hacemos el mismo estudio pero con el headline

```
#Aplicamos la función a los titulares
meta_train_headline = extraer_metadatos(df, 'Headline').add_suffix('_h')

print("Nuevas variables extraídas:")
print(meta_train_headline.head())
```

```
Nuevas variables extraídas:
      longitud_char_h  n_palabras_h  ratio_mayusculas_h  n_exclamaciones_h  \
1052               80             9            0.014493                  0
191                42             7            0.083333                  0
1288               81            12            1.000000                  0
963                39             7            0.090909                  0
1380               34             7            0.071429                  0

      n_interrogaciones_h  n_numeros_h  avg_word_length_h
1052                    0            0           8.000000
191                     0            0           5.250000
1288                    0            1           6.230769
963                     0            0           4.875000
1380                    0            0           4.250000
```

```
#Unimos temporalmente los metadatos con la etiqueta de categoría para graficar
df_analisis = pd.concat([meta_train_headline, df['Category']], axis=1)
metricas = ['ratio_mayusculas_h', 'n_exclamaciones_h', 'n_numeros_h', 'avg_word_length_h', 'n_interrogaciones_h', 'n_palabr

#Grafico
fig, axes = plt.subplots(4, 2, figsize=(15, 12))
fig.suptitle('Comparativa de Metadatos: Noticias Verdaderas vs Falsas', fontsize=16)

for i, metrica in enumerate(metricas):
    row = i // 2
    col = i % 2
    sns.boxplot(x='Category', y=metrica, data=df_analisis, ax=axes[row, col], palette='Set2')
    axes[row, col].set_title(f'Distribución de: {metrica}')
    axes[row, col].set_xlabel('Categoría')
    axes[row, col].set_ylabel('Valor')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

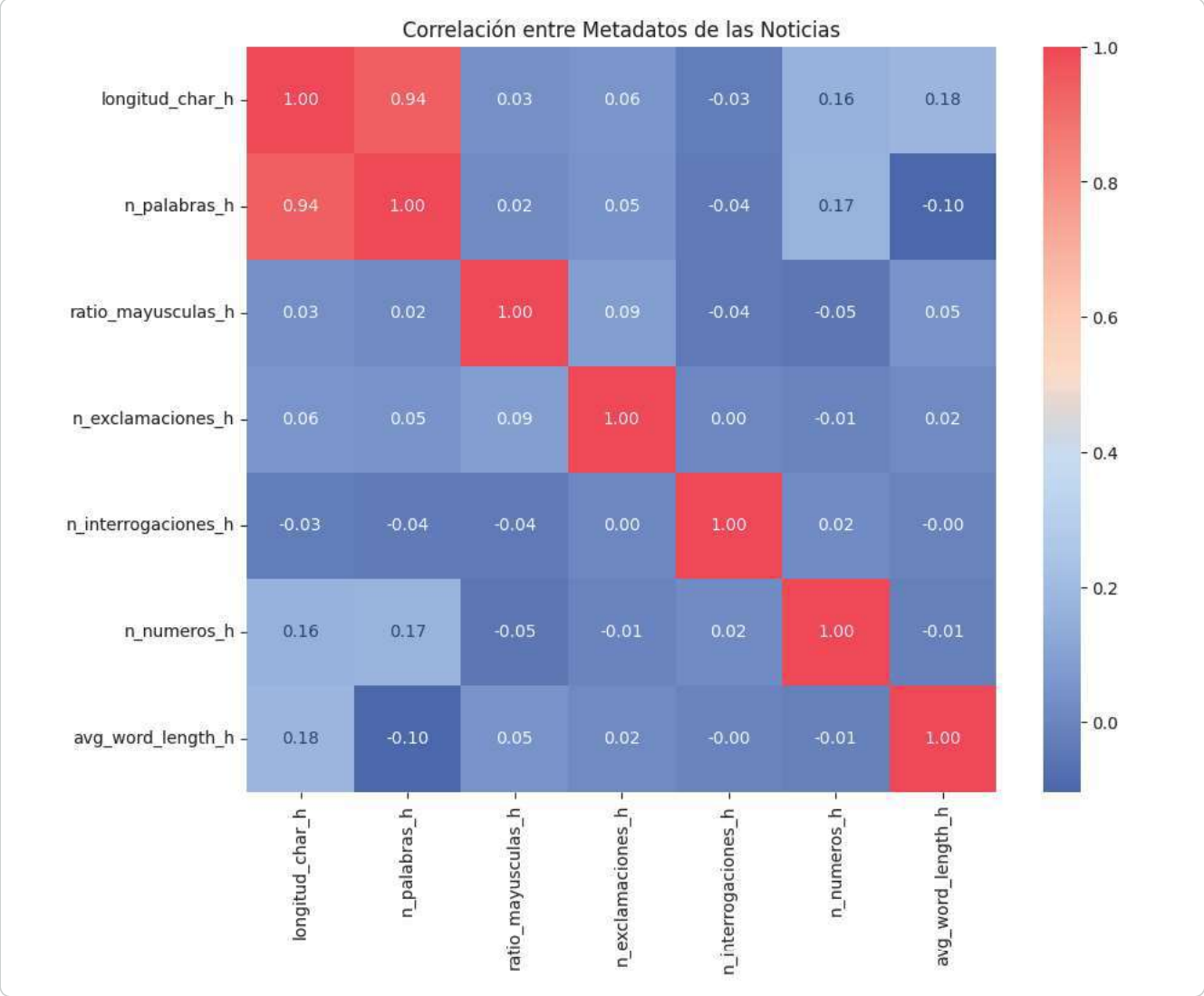
Mostrar salida oculta

```
#Media
resumen_metadatos = df_analisis.groupby('Category').mean()

print("--- VALORES MEDIOS POR CATEGORÍA ---")
display(resumen_metadatos)
```

```
--- VALORES MEDIOS POR CATEGORÍA ---
```

Category	longitud_char_h	n_palabras_h	ratio_mayusculas_h	n_exclamaciones_h	n_interrogaciones_h	n_numeros_h	avg_word_l
Fake	76.469314	12.666065	0.200665	0.028881	0.016245	0.146209	5
True	67.772581	11.154839	0.086583	0.009677	0.062903	0.170968	5

```
#Analizamos si hay correlaciones entre estos datos.
plt.figure(figsize=(10, 8))
sns.heatmap(df_analisis.drop('Category', axis=1).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlación entre Metadatos de las Noticias')
plt.show()
```



Al unir los datos para la clasificación, se eliminará la variable correspondiente a la longitud de caracteres, ya que presenta una correlación perfecta con el número de palabras. Además, esta variable es redundante, puesto que puede derivarse a partir del número de palabras y la longitud media de las mismas, lo que permite calcular la longitud total de caracteres.

Normalización de datos

Voy a normalizar los metadatos que se encuentran en escalas muy distintas y voy a unificar todos los datos (los metadatos, emociones, sentimientos, etc) en un unico dataframe. Ademas, las variables categoricas se tendran que codificar

```
#Hay que juntar el dataframe normal, con los metadatos normalizados, las emociones y los sentimientos y luego ya tengo que
df_total = pd.concat([
    df.drop(columns=['Topic']).reset_index(drop=True),
    meta_train_text.drop(columns=['longitud_char']).reset_index(drop=True),
    meta_train_headline.drop(columns=['longitud_char_h']).reset_index(drop=True),
    resultados_headline.reset_index(drop=True),
    resultados_text.reset_index(drop=True)
], axis=1)

print(df_total.columns.tolist())
print(df_total)
```

```
                                                  Link  n_palabras  \
0     https://www.wradio.com.co/Noticia/actualidad/v...         120
1     http://www.eluniversal.com.mx/espectaculos/far...         329
2     Perma | BIL GATES DETENIDO POR TERRORISMO... -...         558
3     http://www.eluniversal.com.mx/articulo/cultura...         281
4     https://elpais.com/internacional/2020-11-07/vi...        1459
...                                                 ...         ...
1169  https://www.milenio.com/politica/comunidad/cor...         416
1170  https://aristeguinoticias.com/0712/kiosko/cona...         191
1171  https://www.publimetro.cl/cl/estilo-vida/2017/...         285
1172  https://www.eluniversal.com.mx/nacion/amlo-vei...         366
1173  https://www.forbes.com.mx/lozano-regresa-a-sus...         246

      ratio_mayusculas  n_exclamaciones  n_interrogaciones  n_numeros  ...  \
0             0.031667                0                  0          1  ...
1             0.019631                0                  0          0  ...
2             0.031073                0                  0         12  ...
3             0.032975                0                  0          0  ...
4             0.032512                0                  1         15  ...
...                ...              ...                ...        ...  ...
1169          0.030056                0                  0          6  ...
1170          0.040462                0                  0          0  ...
1171          0.036633                0                  1          5  ...
1172          0.037748                1                  0          0  ...
1173          0.071664                0                  0          0  ...

           POS    others       joy   sadness     anger  surprise   disgust  \
0     0.034699  0.966371  0.000357  0.011886  0.005529  0.005283  0.001850
1     0.042564  0.136708  0.004682  0.036881  0.571744  0.040972  0.178877
2     0.095301  0.892426  0.005547  0.010216  0.041265  0.021580  0.015558
3     0.009940  0.945118  0.001013  0.009818  0.029973  0.003181  0.007828
4     0.017108  0.367749  0.003688  0.252809  0.320953  0.001510  0.051601
...        ...       ...       ...       ...       ...       ...       ...
1169  0.050420  0.839911  0.002923  0.041775  0.082627  0.006369  0.017822
1170  0.031470  0.960068  0.002136  0.007480  0.025028  0.001948  0.002489
1171  0.427824  0.990537  0.002695  0.003138  0.001913  0.001144  0.000320
1172  0.132734  0.925678  0.004599  0.020593  0.043818  0.001502  0.003207
1173  0.151568  0.992080  0.003425  0.002290  0.000530  0.001161  0.000278

          fear  polaridad_final  emocion_final
0     0.008723              NEG         others
1     0.030136              NEG          anger
2     0.013408              NEU         others
3     0.003069              NEG         others
```

```
#Quito del df_total polaridad final y emocion final porque al final son una interpretacion del resto de columnas
df_total = df_total.drop(columns=['polaridad_final','emocion_final','polaridad_final_h','emocion_final_h'])
df_total_backup=df_total.copy()
```

```
#Ahora voy a codificar las variables categoricas
df_total['Category'] = df_total['Category'].replace({'True': 1, 'Fake': 0})

#Pasamos source a dummies
df_total = pd.get_dummies(df_total, columns=['Source'], prefix='source', dtype=int)
print(df_total)
```

```
      source_tuexpertomovil  source_twitter  source_twitter - Calderon  \
0                         0               0                          0
1                         0               0                          0
2                         0               0                          0
3                         0               0                          0
4                         0               0                          0
...                     ...             ...                        ...
1169                      0               0                          0
1170                      0               0                          0
1171                      0               0                          0
1172                      0               0                          0
1173                      0               0                          0

      source_twitter -Ricardo Aleman  source_twitter-Rafael Ramirez  \
0                                  0                              0
1                                  0                              0
2                                  0                              0
3                                  0                              0
4                                  0                              0
...                              ...                            ...
1169                               0                              0
1170                               0                              0
1171                               0                              0
1172                               0                              0
1173                               0                              0
```

Modelo embeddings

Embeddings

```
#Cargamos un modelo ligero pero potente para procesar las palabras
model_llm = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')
df_total['full_content'] = df_total['Headline'] + " " + df_total['Text']
embeddings_llm = model_llm.encode(df_total['full_content'].tolist(), show_progress_bar=True)

#Convertimos a DataFrame para unirlo
df_embeddings = pd.DataFrame(embeddings_llm, index=df_total.index)
df_embeddings.columns = [f'emb_llm_{i}' for i in range(df_embeddings.shape[1])]
```

```
Loading weights: 100%                                                    199/199 [00:00<00:00, 735.86it/s]

Batches: 100%                                                          37/37 [00:05<00:00, 11.64it/s]
```

Ahora ya tengo el texto y el headline en version "numerica" para que el modelo de clasificación lo pueda entender

```
#Ahora lo uno todo en un unico dataframe
X = pd.concat([df_total.drop(['Text', 'Headline', 'Link', 'Category','full_content'], axis=1), df_embeddings], axis=1)
y = df_total['Category']
```

```
#Convierto los NaN en 0s para que el modelo pueda trabajar
print(X.isnull().sum())
```

```
n_palabras           0
ratio_mayusculas     0
n_exclamaciones      0
n_interrogaciones    0
n_numeros            0
                    ..
emb_llm_379          0
emb_llm_380          0
emb_llm_381          0
emb_llm_382          0
emb_llm_383          0
Length: 712, dtype: int64
```

```
#Como hay algunas filas con NaN voy a eliminar esas filas en vez de rellenarlas con 0 para que el modelo no aprenda patrone
X = X.dropna()
y = y.loc[X.index]
```

```
scaler = MinMaxScaler()
# Escalamos todo el bloque X
X_escalado = pd.DataFrame(scaler.fit_transform(X),
                          columns=X.columns,
                          index=X.index)
```

Ahora tengo que poner el conjunto de datos de test en el mismo formato que los datos de entrenamiento para que el random forest lo pueda identificar. Es muy importante que todas las columnas sean iguales. Es decir, si la 48 es la de sentimiento en el train en el test tambien debe serlo para que el random forest lo pueda identificar bien. Ademas, a la hora de escalarlo tengo que hacer solo scaler_transform, sin el fit, para que el modelo no vuelva a intentar detectar max y minimos y lo escale de la misma forma que los datos de train.

```
#Preparo los datos para el test. Es decir, lo dejo en el mismo formato que estaba el train; con sentimientos, metadatos, er

#Me aseguro de que estan en el formato adecuado
test['Headline'] = test['Headline'].astype(str).fillna('')
test['Text'] = test['Text'].astype(str).fillna('')

#Analisis de sentimientos
resultados_test_Headline = test['Headline'].apply(analizar_pysentimiento).apply(pd.Series).add_suffix('_h')
resultados_test_text = test['Text'].apply(analizar_pysentimiento).apply(pd.Series)

#Metadatos
meta_test_text = extraer_metadatos(test, 'Text')
meta_test_headline = extraer_metadatos(test, 'Headline').add_suffix('_h')

#Unimos todos los datos
df_test = pd.concat([test, meta_test_text, meta_test_headline, resultados_test_Headline, resultados_test_text], axis=1)

#Codificamos
df_test['Category'] = df_test['Category'].replace({
    'TRUE': 1,
    'True': 1,
    True: 1,
    'Fake': 0,
    False: 0
})
df_test = pd.get_dummies(df_test, columns=['Source'], prefix='source', dtype=int)

#Embeddings
df_test['full_content'] = df_test['Headline'].astype(str) + " " + df_test['Text'].astype(str)
embeddings_llm_test = model_llm.encode(df_test['full_content'].tolist(), show_progress_bar=True)
df_embeddings_test = pd.DataFrame(embeddings_llm_test, index=df_test.index)
df_embeddings_test.columns = [f'emb_llm_{i}' for i in range(df_embeddings_test.shape[1])]

#Unificamos y dividimos en X e Y
X_test = pd.concat([df_test.drop(['Text', 'Headline', 'Link', 'Category','full_content','polaridad_final','emocion_final',
y_test = df_test['Category']

#Quitamos los datos que sean NaN
X_test = X_test.dropna()
y_test = y_test.loc[X_test.index]

#Me aseguro de que X test tiene exactamente las mismas columas que x train escalado
missing_cols = set(X_escalado.columns) - set(X_test.columns)
for c in missing_cols:
    X_test[c] = 0
X_test = X_test[X_escalado.columns]

X_test_escalado = pd.DataFrame(scaler.transform(X_test),
                               columns=X_test.columns,
                               index=X_test.index)
```

Mostrar salida oculta

```
#Compruebo que ambos bloques son iguales
test = X_test_escalado.columns.tolist()
print(test)
train=X_escalado.columns.tolist()
print(train)
```

```
['n_palabras', 'ratio_mayusculas', 'n_exclamaciones', 'n_interrogaciones', 'n_numeros', 'avg_word_length', 'n_palabras_h', '
['n_palabras', 'ratio_mayusculas', 'n_exclamaciones', 'n_interrogaciones', 'n_numeros', 'avg_word_length', 'n_palabras_h', '
```

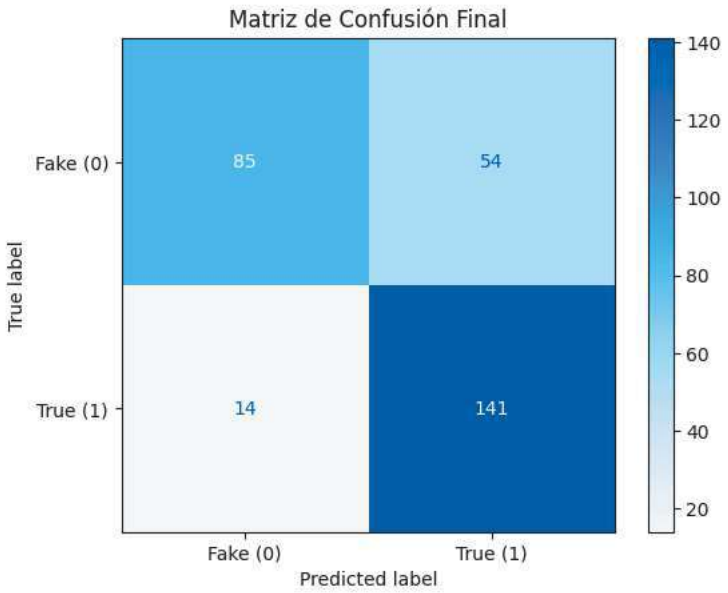
Random forest

```
#Hago un random forest con los datos de entrenamiento
y = y.astype(int)
rf_model = RandomForestClassifier(n_estimators=200, random_state=42,class_weight='balanced')
rf_model.fit(X_escalado, y)
```

```
RandomForestClassifier(class_weight='balanced', n_estimators=200,
                       random_state=42)
```

```
y_pred_test = rf_model.predict(X_test_escalado)
y_test = y_test.astype(int)
print(classification_report(y_test, y_pred_test))

#Creo la matrix de confusion
cm = confusion_matrix(y_test, y_pred_test)

# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.86      0.61      0.71       139
           1       0.72      0.91      0.81       155

    accuracy                           0.77       294
   macro avg       0.79      0.76      0.76       294
weighted avg       0.79      0.77      0.76       294

<Figure size 800x600 with 0 Axes>
```



Vemos que predice peor los 0 que los 1 y, en este caso, es mas importante ser capaz de detectar los 0 por lo que vamos a tratar de mejorar eso.

```
#Voy a analizar la correlación entre todas las variables que estoy introduciendo al random forest
data = pd.concat([X_escalado], axis=1)
data.corr().round(3)
```

Mostrar salida oculta

```
#Calculamos la matriz de correlación
corr_matrix = data.corr()
np.fill_diagonal(corr_matrix.values, 0)
corr_pairs = corr_matrix.unstack()
corr_pairs_sorted = corr_pairs.reindex(corr_pairs.abs().sort_values(ascending=False).index)

#Mostramos las top 10 correlaciones más fuertes
top_X = 10
print(corr_pairs_sorted.head(top_X))
```

```
NEU_h    NEG_h     -0.828890
NEG_h    NEU_h     -0.828890
anger    disgust    0.789622
disgust  anger      0.789622
NEU      NEG       -0.787857
NEG      NEU       -0.787857
others   anger     -0.773607
```

```
anger    others    -0.773607
NEG      POS       -0.714949
POS      NEG       -0.714949
dtype: float64
```

```
# He visto que hay correlaciones altas entre varias variables. Voy a intentar eliminarlas para ver si eso mejora el modelo.
cols_a_eliminar = set()
umbral = 0.6
df_corr=data.corr()


for i in range(len(df_corr.columns)):
    for j in range(i):
        if df_corr.iloc[i, j] > umbral:
            colname = df_corr.columns[i]
            cols_a_eliminar.add(colname)

print(f"Se eliminarán {len(cols_a_eliminar)} columnas por alta correlación:")
print(cols_a_eliminar)

#Creamos un nuevo DataFrame sin las columnas redundantes
X_limpia = X_escalado.drop(columns=cols_a_eliminar)

print(f"Número de columnas antes: {X_escalado.shape[1]}")
print(f"Número de columnas después: {X_limpia.shape[1]}")
```

```
Se eliminarán 4 columnas por alta correlación:
{'NEG', 'others', 'disgust', 'disgust_h'}
Número de columnas antes: 712
Número de columnas después: 708
```

```
#Vuelvo a hacer el random forest pero con la X sin altas correlaciones
rf_model = RandomForestClassifier(n_estimators=200, random_state=42)
rf_model.fit(X_limpia, y)
```

```
▾              RandomForestClassifier
RandomForestClassifier(n_estimators=200, random_state=42)
```

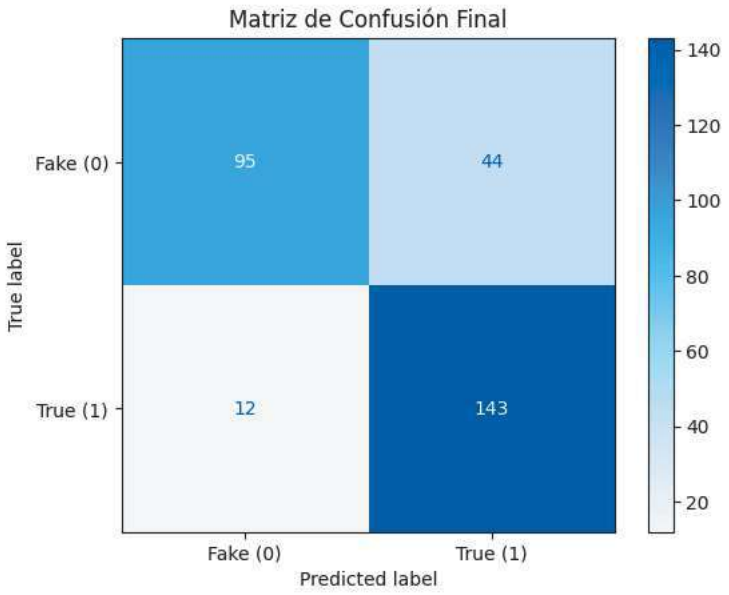
```
#quitamos del test las columnas eliminadas
X_test_escalado_limpia = X_test_escalado.drop(columns=['disgust_h', 'others', 'NEG', 'disgust'])
```

```
y_pred_test = rf_model.predict(X_test_escalado_limpia)
y_test = y_test.astype(int)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.89      0.68      0.77       139
           1       0.76      0.92      0.84       155

    accuracy                           0.81       294
   macro avg       0.83      0.80      0.80       294
weighted avg       0.82      0.81      0.81       294
```

```
#Creamos la matriz de confusión
cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

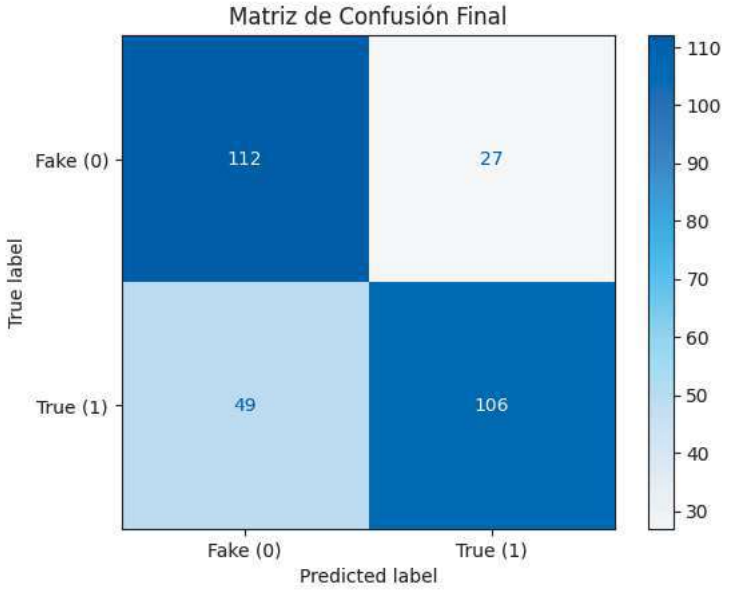
```
<Figure size 800x600 with 0 Axes>
```



Podemos ver que eliminando las variablas con mas correlacion mejora el modelo

```
#Intento subir el umbral para que mejore la detección de fake news
y_proba = rf_model.predict_proba(X_test_escalado_limpia)[:, 1]
y_pred_adj = (y_proba > 0.55).astype(int)

print(classification_report(y_test, y_pred_adj))

cm = confusion_matrix(y_test, y_pred_adj)

#La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

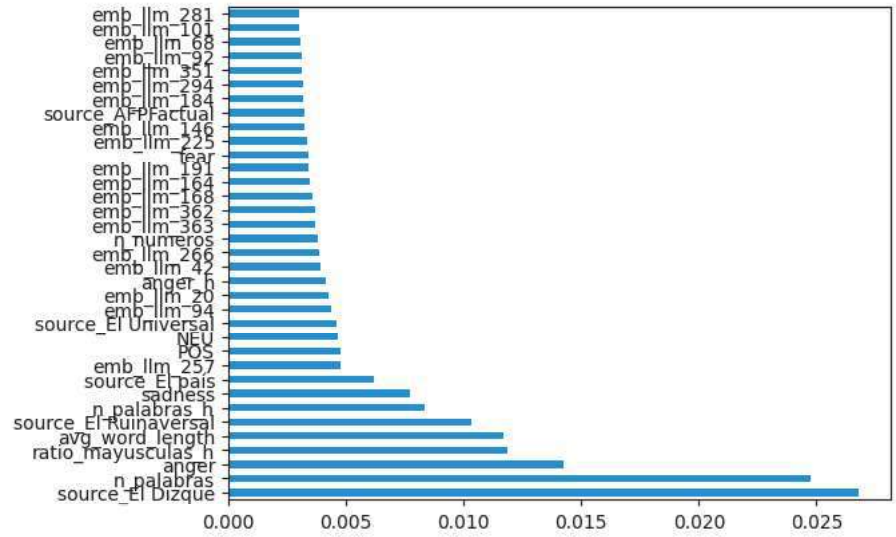
```
              precision    recall  f1-score   support

           0       0.70      0.81      0.75       139
           1       0.80      0.68      0.74       155

    accuracy                           0.74       294
   macro avg       0.75      0.74      0.74       294
weighted avg       0.75      0.74      0.74       294

<Figure size 800x600 with 0 Axes>
```



Bajando un poco el umbral empeora un poco el modelo pero captura mejora las noticias falsas.

```
#Probamos un modelo en el que solo coge las 35 primeras
importances = pd.Series(rf_model.feature_importances_, index=X_limpia.columns)
importances.nlargest(35).plot(kind='barh')
```

```
<Axes: >
```

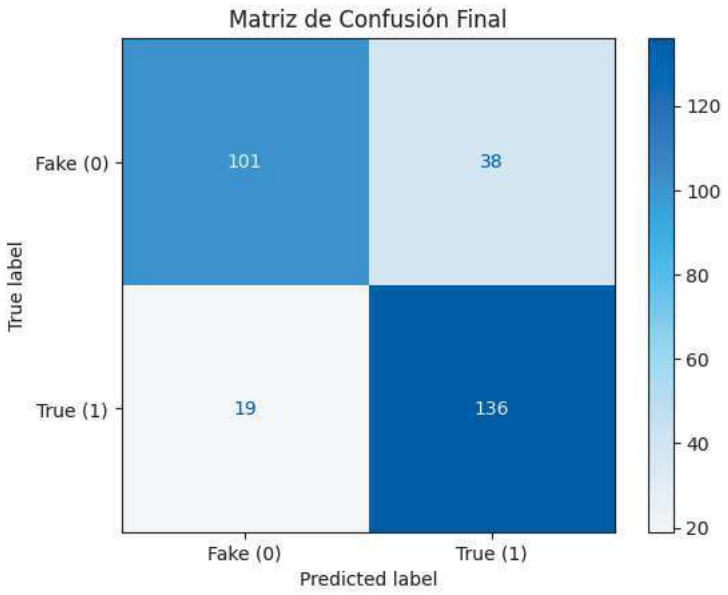



```
#Pruebo un modelo con unicamente estas 35 variables mas importantes
#Obtengo los nombres da las variables mas importantes
top_35_features = importances.nlargest(35).index.tolist()

X_top35 = X_limpia[top_35_features]
X_top35_test=X_test_escalado_limpia[top_35_features]

rf_model = RandomForestClassifier(n_estimators=200, random_state=42)
rf_model.fit(X_top35, y)

y_pred_test = rf_model.predict(X_top35_test)
y_test = y_test.astype(int)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.84      0.73      0.78       139
           1       0.78      0.88      0.83       155

    accuracy                           0.81       294
   macro avg       0.81      0.80      0.80       294
weighted avg       0.81      0.81      0.80       294
```

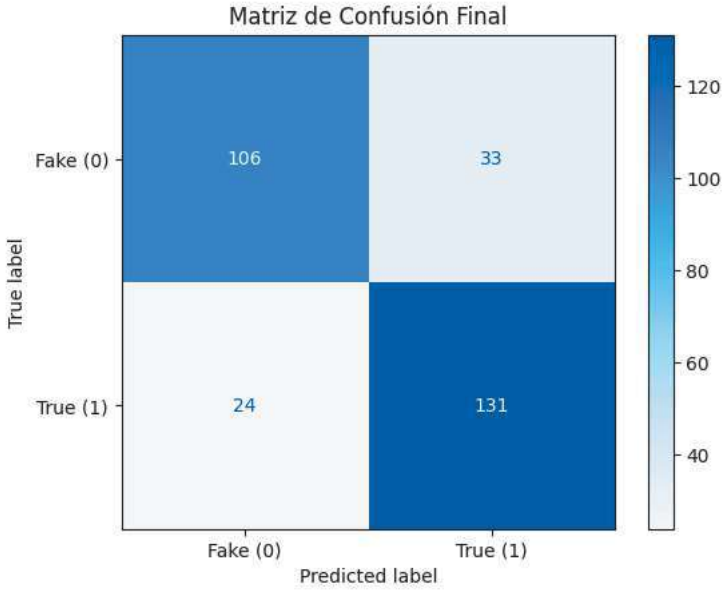
```
# Creamos la matriz de confusión
cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```




```
#Intento subir el umbral para que mejore la detección de fake news
y_proba = rf_model.predict_proba(X_top35_test)[:, 1]
y_pred_adj = (y_proba > 0.55).astype(int)

print(classification_report(y_test, y_pred_adj))

cm = confusion_matrix(y_test, y_pred_adj)

# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.82      0.76      0.79       139
           1       0.80      0.85      0.82       155

    accuracy                           0.81       294
   macro avg       0.81      0.80      0.80       294
weighted avg       0.81      0.81      0.81       294

<Figure size 800x600 with 0 Axes>
```



De momento, nos quedamos con el modelo entero quitando aquellas correlaciones mas altas

Modelo basado solo en embeddings

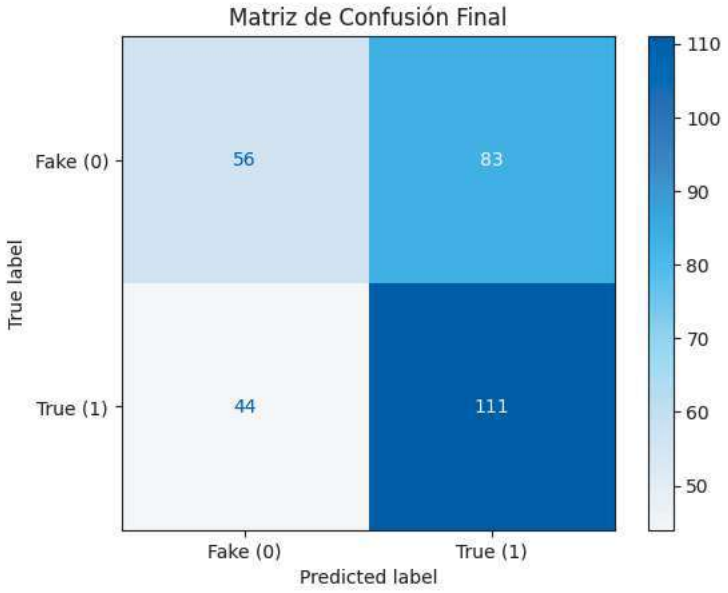
```
#Preparo tanto los datos de entrenamiento como los de test

#Los datos de entrenamiento
X2 = pd.concat([df_total.drop(['Text', 'Headline', 'Link', 'Category','full_content','full_content','n_palabras', 'ratio_ma

#Los datos de test
X2_test = X_test.loc[:, X_test.columns.str.startswith('emb')]
```

```
#Random forest
rf_model = RandomForestClassifier(n_estimators=200, random_state=42,class_weight='balanced')
rf_model.fit(X2, y)
```

```
RandomForestClassifier(class_weight='balanced', n_estimators=200,
                       random_state=42)
```

```
y_pred_test = rf_model.predict(X2_test)
y_test = y_test.astype(int)
print(classification_report(y_test, y_pred_test))

# Creamos la matriz de confusion
cm = confusion_matrix(y_test, y_pred_test)

# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.56      0.40      0.47       139
           1       0.57      0.72      0.64       155

    accuracy                           0.57       294
   macro avg       0.57      0.56      0.55       294
weighted avg       0.57      0.57      0.56       294

<Figure size 800x600 with 0 Axes>
```



PCA Embeddings + Random forest

```
#Hago PCA de los embeddings para reducir su dimensionalidad sin llegar a perder informacion
pca = PCA(n_components=50, random_state=42)

emb_train_pca = pca.fit_transform(df_embeddings)
emb_test_pca  = pca.transform(df_embeddings_test)
```

```
#Añado los embeddings PCA a X (quitando embeddings anteriores)

emb_cols_orig = [c for c in X.columns if c.startswith('emb_')]
X_no_emb = X.drop(columns=emb_cols_orig)
X_test_no_emb = X_test.drop(columns=emb_cols_orig)


emb_cols_pca = [f'emb_pca_{i}' for i in range(50)]

X_pca = pd.concat([
    X_no_emb.reset_index(drop=True),
    pd.DataFrame(emb_train_pca, columns=emb_cols_pca)
], axis=1)

X_test_pca = pd.concat([
    X_test_no_emb.reset_index(drop=True),
    pd.DataFrame(emb_test_pca, columns=emb_cols_pca)
], axis=1)

#COmprobamos si las columnas estan alineadas
print_train=X_pca.columns.tolist()
print(print_train)
print_test=X_test_pca.columns.tolist()
print(print_test)
```

```
['n_palabras', 'ratio_mayusculas', 'n_exclamaciones', 'n_interrogaciones', 'n_numeros', 'avg_word_length', 'n_palabras_h', '
['n_palabras', 'ratio_mayusculas', 'n_exclamaciones', 'n_interrogaciones', 'n_numeros', 'avg_word_length', 'n_palabras_h', '
```

```
#Hay una fila en el dataset que tienen NaN asi que la elimino
X_pca = X_pca.dropna()
#Ajusto la columan de y para que tambien se eliminen las filas que se han eliminado en las x
y = y.reset_index(drop=True)
```

```
#Hago el random forest
rf = RandomForestClassifier(
    n_estimators=400,
    max_depth=15,
    min_samples_leaf=5,
    max_features=0.3,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)
rf.fit(X_pca, y)
```

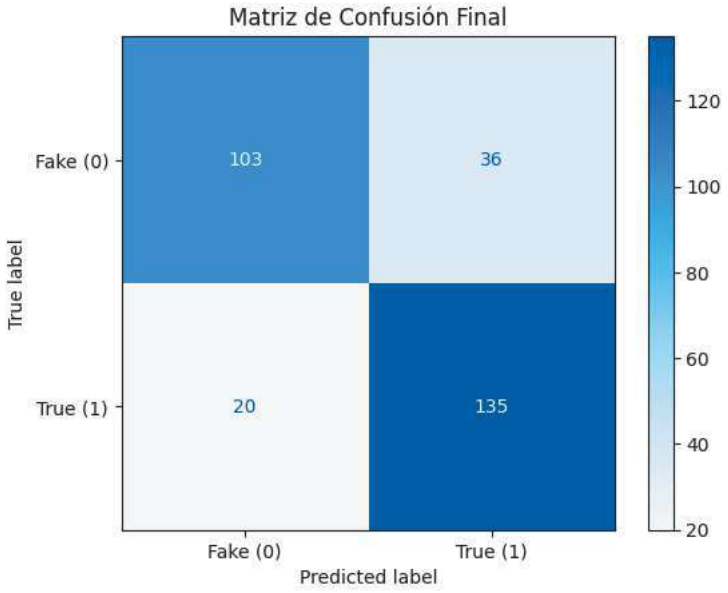
```
RandomForestClassifier(class_weight='balanced', max_depth=15, max_features=0.3,
                       min_samples_leaf=5, n_estimators=400, n_jobs=-1,
                       random_state=42)
```

```
#Evaluo
y_pred_pca = rf.predict(X_test_pca)

print(classification_report(
    y_test,
    y_pred_pca,
    target_names=['Fake (0)', 'True (1)']
))

confusion_matrix(y_test, y_pred_pca)
```

```
              precision    recall  f1-score   support

    Fake (0)       0.84      0.74      0.79       139
    True (1)       0.79      0.87      0.83       155

    accuracy                           0.81       294
   macro avg       0.81      0.81      0.81       294
weighted avg       0.81      0.81      0.81       294

array([[103,  36],
       [ 20, 135]])
```

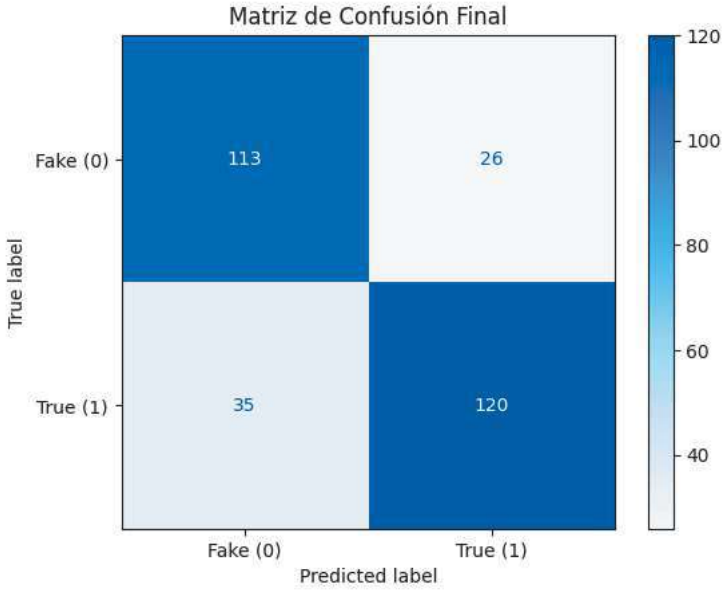
```
# Creamos la matriz de confusión
cm = confusion_matrix(y_test, y_pred_pca)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```




```
#Intento subir el umbral para que mejore la detección de fake news
y_proba = rf.predict_proba(X_test_pca)[:, 1]
y_pred_adj = (y_proba > 0.6).astype(int)

print(classification_report(y_test, y_pred_adj))

cm = confusion_matrix(y_test, y_pred_adj)

# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.76      0.81      0.79       139
           1       0.82      0.77      0.80       155

    accuracy                           0.79       294
   macro avg       0.79      0.79      0.79       294
weighted avg       0.79      0.79      0.79       294

<Figure size 800x600 with 0 Axes>
```



De momento, el modelo habiendo quitado las variables con mayor correlación sigue siendo el mejor.

Vamor a probar a hacer lo mismo aqui.

```
cols_a_eliminar = set()

umbral = 0.6
df_corr= X_pca.corr()

for i in range(len(df_corr.columns)):
    for j in range(i):
        if df_corr.iloc[i, j] > umbral:
            colname = df_corr.columns[i]
            cols_a_eliminar.add(colname)

print(f"Se eliminarán {len(cols_a_eliminar)} columnas por alta correlación:")
print(cols_a_eliminar)

#Creamos un nuevo DataFrame sin las columnas redundantes
X_limpia_pca = X_pca.drop(columns=cols_a_eliminar)

print(f"Número de columnas antes: {X_escalado.shape[1]}")
print(f"Número de columnas después: {X_limpia.shape[1]}")
```

```
Se eliminarán 4 columnas por alta correlación:
{'NEG', 'others', 'disgust', 'disgust_h'}
Número de columnas antes: 712
Número de columnas después: 708
```

```
#Vuelvo a hacer el random forest pero con la X sin altas correlaciones
rf_model = RandomForestClassifier(n_estimators=200, random_state=42)
rf_model.fit(X_limpia_pca, y)
```

```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=200, random_state=42)
```

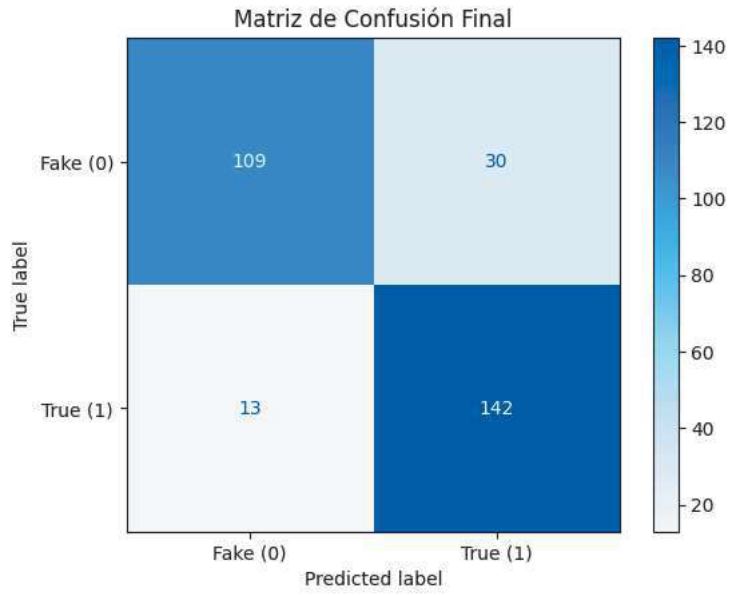
```
#quitamos de la Y las columnas eliminadas
X_test_limpia_pca = X_test_pca.drop(columns=['disgust', 'NEG', 'disgust_h', 'others'])
```

```
y_pred_test = rf_model.predict(X_test_limpia_pca)
y_test = y_test.astype(int)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.89      0.78      0.84       139
           1       0.83      0.92      0.87       155

    accuracy                           0.85       294
   macro avg       0.86      0.85      0.85       294
weighted avg       0.86      0.85      0.85       294
```

```
# Creamos la matriz de confusión
cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

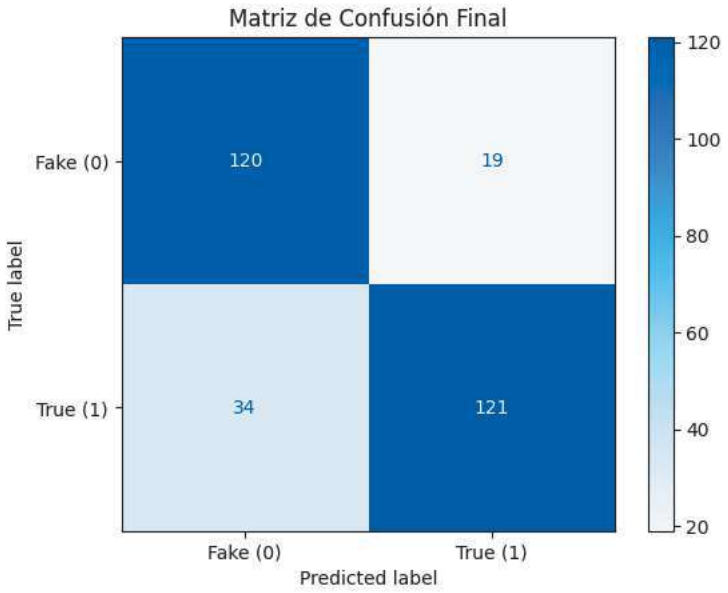
```
<Figure size 800x600 with 0 Axes>
```



Parece que puede captar mejor las noticias falsas

```
#Intento subir el umbral para que mejore la detección de fake news
y_proba = rf_model.predict_proba(X_test_limpia_pca)[:, 1]
y_pred_adj = (y_proba > 0.55).astype(int)

print(classification_report(y_test, y_pred_adj))

cm = confusion_matrix(y_test, y_pred_adj)

# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       139
           1       0.86      0.78      0.82       155

    accuracy                           0.82       294
   macro avg       0.82      0.82      0.82       294
weighted avg       0.82      0.82      0.82       294

<Figure size 800x600 with 0 Axes>
```



Logistic regression + random forest

```
#Ahora hago un logistic regression para los embeddings (lo hago sobre el PCA de los embeddings)
scaler = StandardScaler()
emb_train_scaled = scaler.fit_transform(emb_train_pca)
emb_test_scaled = scaler.transform(emb_test_pca)

emb_train_scaled = pd.DataFrame(emb_train_scaled, columns=emb_cols_pca)
```

```
emb_train_scaled = emb_train_scaled.loc[X.index]

#Logistic Regression con embeddings
lr_emb = LogisticRegression(max_iter=1000, random_state=42)
lr_emb.fit(emb_train_scaled, y)

#Probabilidades
lr_train_probs = lr_emb.predict_proba(emb_train_scaled)[:, 1]
lr_test_probs = lr_emb.predict_proba(emb_test_scaled)[:, 1]
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names,
  warnings.warn(
```

```
#Random Forest con metadatos + sentimientos
rf_meta = RandomForestClassifier(n_estimators=200, random_state=42)
rf_meta.fit(X_no_emb, y)

# Probabilidades
rf_train_probs = rf_meta.predict_proba(X_no_emb)[:, 1]
rf_test_probs = rf_meta.predict_proba(X_test_no_emb)[:, 1]  # test completo
```

```
#Junto ambas
stack_train = np.column_stack([lr_train_probs, rf_train_probs])
stack_test = np.column_stack([lr_test_probs, rf_test_probs])

meta_model = LogisticRegression(max_iter=1000, random_state=42)
meta_model.fit(stack_train, y)
```

```
LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

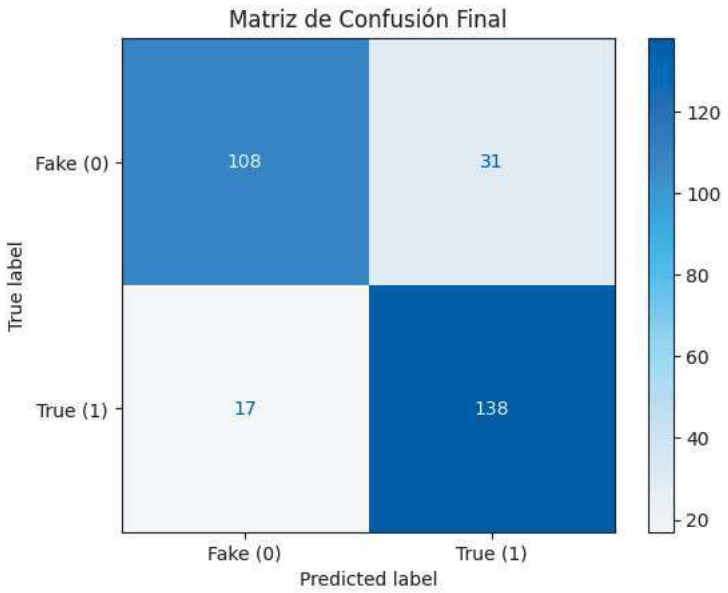
```
#Predicciones finales
final_train_preds = meta_model.predict(stack_train)
final_test_preds = meta_model.predict(stack_test)
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, final_test_preds))
```

```
              precision    recall  f1-score   support

           0       0.86      0.78      0.82       139
           1       0.82      0.89      0.85       155

    accuracy                           0.84       294
   macro avg       0.84      0.83      0.84       294
weighted avg       0.84      0.84      0.84       294
```

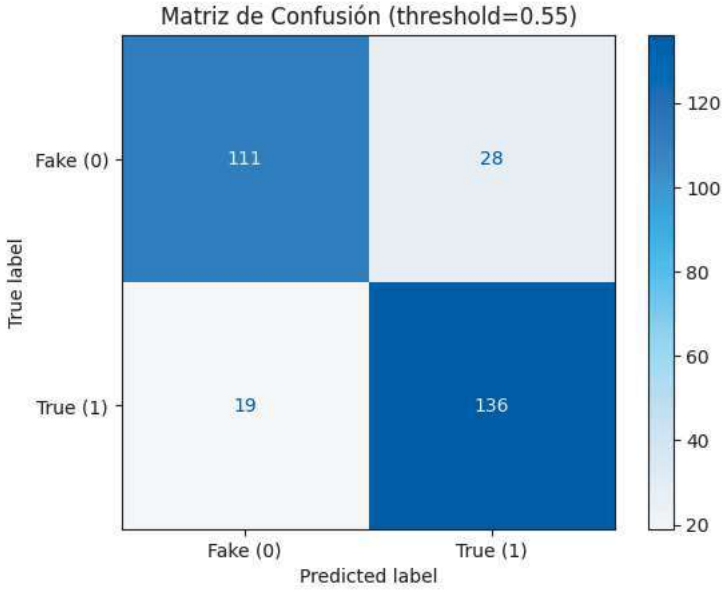
```
cm = confusion_matrix(y_test, final_test_preds)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



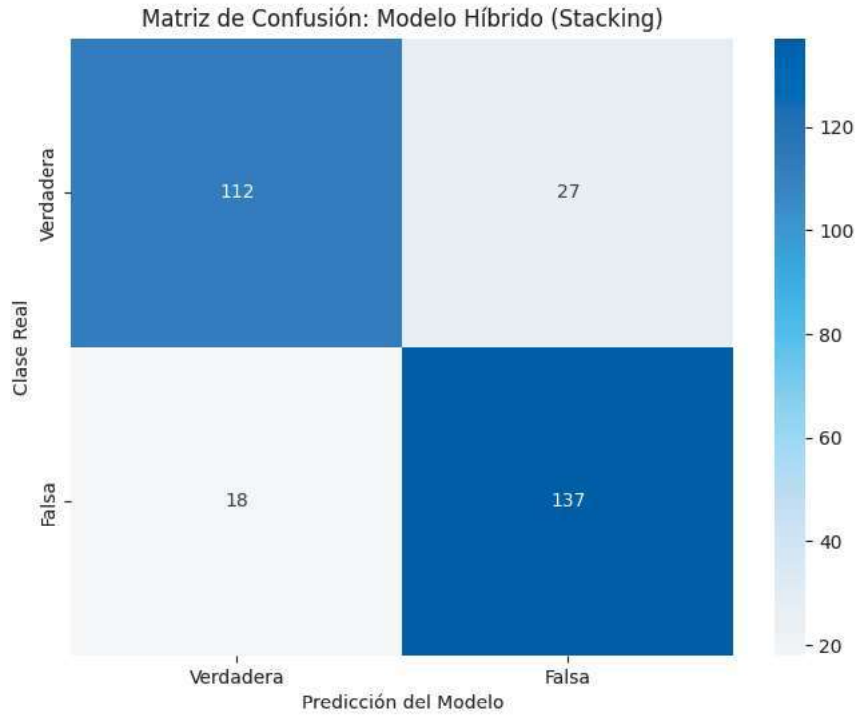

```
y_proba = meta_model.predict_proba(stack_test)[:, 1]
# Ajustar el umbral
threshold = 0.55
y_pred_adj = (y_proba > threshold).astype(int)

# Reporte
print(classification_report(y_test, y_pred_adj))

# Matriz de confusión
cm = confusion_matrix(y_test, y_pred_adj)

import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title(f'Matriz de Confusión (threshold={threshold})')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.85      0.80      0.83       139
           1       0.83      0.88      0.85       155

    accuracy                           0.84       294
   macro avg       0.84      0.84      0.84       294
weighted avg       0.84      0.84      0.84       294

<Figure size 800x600 with 0 Axes>
```




```
#Hacemos lo mismo pero con los embeddigns sin PCA
scaler = StandardScaler()
emb_train_scaled = scaler.fit_transform(df_embeddings)
emb_test_scaled = scaler.transform(df_embeddings_test)

emb_train_scaled = pd.DataFrame(emb_train_scaled, columns=df_embeddings.columns, index=df_embeddings.index)
emb_train_scaled = emb_train_scaled.loc[X.index]


lr_emb = LogisticRegression(max_iter=1000, random_state=42)
lr_emb.fit(emb_train_scaled, y)
lr_train_probs = lr_emb.predict_proba(emb_train_scaled)[:, 1]
lr_test_probs = lr_emb.predict_proba(emb_test_scaled)[:, 1]


rf_meta = RandomForestClassifier(n_estimators=200, random_state=42)
rf_meta.fit(X_no_emb, y)
rf_train_probs = rf_meta.predict_proba(X_no_emb)[:, 1]
rf_test_probs = rf_meta.predict_proba(X_test_no_emb)[:, 1]

stack_train = np.column_stack([lr_train_probs, rf_train_probs])
stack_test = np.column_stack([lr_test_probs, rf_test_probs])

meta_model = LogisticRegression(max_iter=1000, random_state=42)
meta_model.fit(stack_train, y)

final_test_preds = meta_model.predict(stack_test)


print(classification_report(y_test, final_test_preds))

# Dibujar la Matriz de Confusión
cm = confusion_matrix(y_test, final_test_preds)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Verdadera', 'Falsa'],
            yticklabels=['Verdadera', 'Falsa'])
plt.title('Matriz de Confusión: Modelo Híbrido (Stacking)')
plt.xlabel('Predicción del Modelo')
plt.ylabel('Clase Real')
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names,
  warnings.warn(
              precision    recall  f1-score   support

           0       0.86      0.81      0.83       139
           1       0.84      0.88      0.86       155

    accuracy                           0.85       294
   macro avg       0.85      0.84      0.85       294
weighted avg       0.85      0.85      0.85       294
```



Pruebo a quitar correlaciones. De momento, el modelo es mejor sin hacer PCA

```
y = y.astype(int)

selector_meta = SelectKBest(score_func=f_classif, k=40)
X_no_emb_top40 = selector_meta.fit_transform(X_no_emb, y)
X_test_no_emb_top40 = selector_meta.transform(X_test_no_emb)

rf_meta_red = RandomForestClassifier(n_estimators=200, random_state=42)
rf_meta_red.fit(X_no_emb_top40, y)

selector_pca = SelectKBest(score_func=f_classif, k=40)
emb_train_pca_aligned = X_pca[emb_cols_pca]
emb_train_pca_top40 = selector_pca.fit_transform(emb_train_pca_aligned, y)
emb_test_pca_aligned = X_test_pca[emb_cols_pca]
emb_test_pca_top40 = selector_pca.transform(emb_test_pca_aligned)

lr_emb_red = LogisticRegression(max_iter=1000, random_state=42)
lr_emb_red.fit(emb_train_pca_top40, y)


rf_probs_red = rf_meta_red.predict_proba(X_no_emb_top40)[:, 1]
lr_probs_red = lr_emb_red.predict_proba(emb_train_pca_top40)[:, 1]

rf_test_probs_red = rf_meta_red.predict_proba(X_test_no_emb_top40)[:, 1]
lr_test_probs_red = lr_emb_red.predict_proba(emb_test_pca_top40)[:, 1]

stack_train_red = np.column_stack([lr_probs_red, rf_probs_red])
stack_test_red = np.column_stack([lr_test_probs_red, rf_test_probs_red])

meta_model_final = LogisticRegression(max_iter=1000, random_state=42)
meta_model_final.fit(stack_train_red, y)

# Predicciones finales sobre el set de TEST
final_preds = meta_model_final.predict(stack_test_red)

# Verifico que y_test esté definido y alineado con stack_test_red
print(classification_report(y_test, final_preds))
```

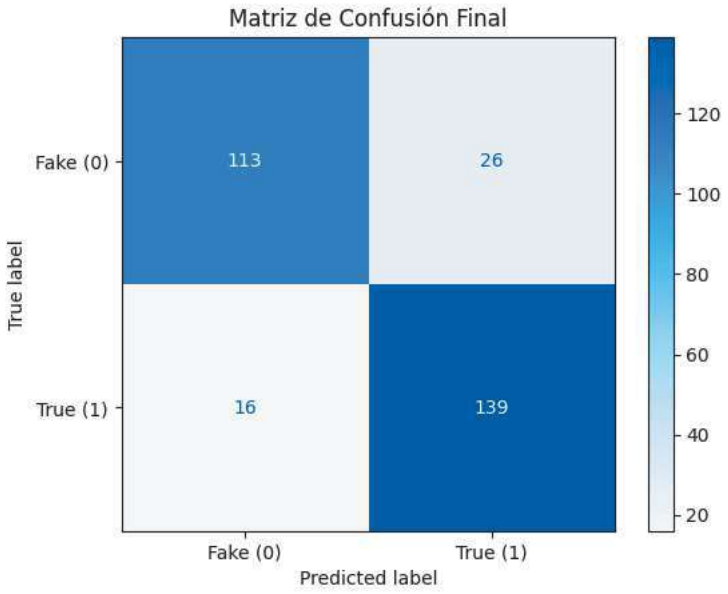
```
              precision    recall  f1-score   support
```

```
              0       0.83      0.76      0.80       139
              1       0.80      0.86      0.83       155

    accuracy                            0.82       294
   macro avg       0.82      0.81      0.81       294
weighted avg       0.82      0.82      0.82       294
```

El modelo no mejora con las x features mas importantes. Por ello pasamos a probar quitando las correlaciones mas elevadas.

```
scaler = StandardScaler()
emb_train_scaled = scaler.fit_transform(emb_train_pca)
emb_test_scaled = scaler.transform(emb_test_pca)

df_emb_train = pd.DataFrame(emb_train_scaled, columns=emb_cols_pca).loc[X.index]
df_emb_test = pd.DataFrame(emb_test_scaled, columns=emb_cols_pca)

X_stack_completo = pd.concat([df_emb_train.reset_index(drop=True), X_no_emb.reset_index(drop=True)], axis=1)
X_test_stack_completo = pd.concat([df_emb_test.reset_index(drop=True), X_test_no_emb.reset_index(drop=True)], axis=1)

cols_a_eliminar = set()
umbral = 0.6
df_corr = X_stack_completo.corr()

for i in range(len(df_corr.columns)):
    for j in range(i):
        if abs(df_corr.iloc[i, j]) > umbral:
            colname = df_corr.columns[i]
            cols_a_eliminar.add(colname)

print(f"Se eliminarán {len(cols_a_eliminar)} columnas por alta correlación (> {umbral})")

X_limpia_2 = X_stack_completo.drop(columns=cols_a_eliminar)
X_test_limpia_2 = X_test_stack_completo.drop(columns=cols_a_eliminar)

cols_emb_final = [c for c in X_limpia_2.columns if c in emb_cols_pca]
cols_meta_final = [c for c in X_limpia_2.columns if c not in emb_cols_pca]

lr_emb = LogisticRegression(max_iter=1000, random_state=42)
lr_emb.fit(X_limpia_2[cols_emb_final], y)

rf_meta = RandomForestClassifier(n_estimators=200, random_state=42)
rf_meta.fit(X_limpia_2[cols_meta_final], y)


lr_train_probs = lr_emb.predict_proba(X_limpia_2[cols_emb_final])[:, 1]
lr_test_probs = lr_emb.predict_proba(X_test_limpia_2[cols_emb_final])[:, 1]
rf_train_probs = rf_meta.predict_proba(X_limpia_2[cols_meta_final])[:, 1]
rf_test_probs = rf_meta.predict_proba(X_test_limpia_2[cols_meta_final])[:, 1]

stack_train = np.column_stack([lr_train_probs, rf_train_probs])
stack_test = np.column_stack([lr_test_probs, rf_test_probs])

meta_model = LogisticRegression(max_iter=1000, random_state=42)
meta_model.fit(stack_train, y)

final_test_preds = meta_model.predict(stack_test)

# Reporte final
y_test = y_test.astype(int)
print(classification_report(y_test, final_test_preds))

cm = confusion_matrix(y_test, final_test_preds)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

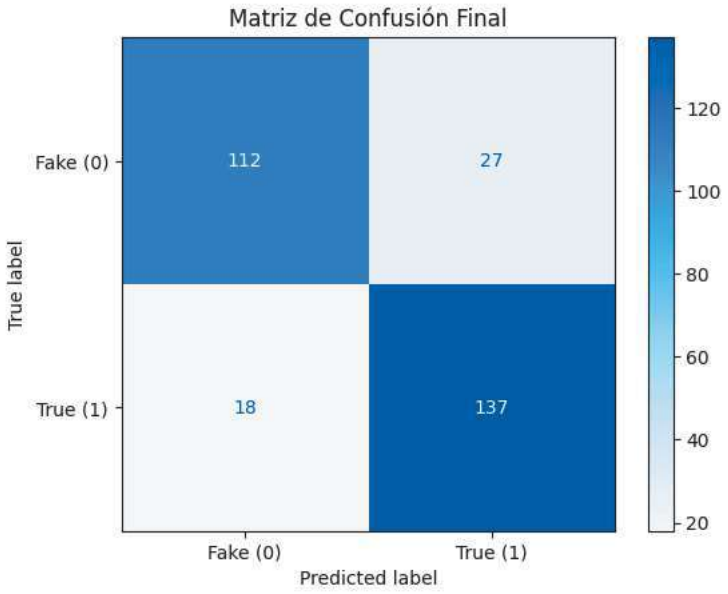
```
Se eliminarán 10 columnas por alta correlación (> 0.6)
              precision    recall  f1-score   support

           0       0.88      0.81      0.84       139
           1       0.84      0.90      0.87       155

    accuracy                           0.86       294
   macro avg       0.86      0.85      0.86       294
weighted avg       0.86      0.86      0.86       294

<Figure size 800x600 with 0 Axes>
```



Como el modelo quitando correlaciones mejora, lo probamos tambien sin PCA

```
scaler = StandardScaler()
emb_train_scaled = scaler.fit_transform(df_embeddings)
emb_test_scaled = scaler.transform(df_embeddings_test)

df_emb_train = pd.DataFrame(emb_train_scaled, columns=df_embeddings.columns).loc[X.index]
df_emb_test = pd.DataFrame(emb_test_scaled, columns=df_embeddings_test.columns)

X_stack_completo = pd.concat([df_emb_train.reset_index(drop=True), X_no_emb.reset_index(drop=True)], axis=1)
X_test_stack_completo = pd.concat([df_emb_test.reset_index(drop=True), X_test_no_emb.reset_index(drop=True)], axis=1)

cols_a_eliminar = set()
umbral = 0.6
df_corr = X_stack_completo.corr()

for i in range(len(df_corr.columns)):
    for j in range(i):
        if abs(df_corr.iloc[i, j]) > umbral:
            colname = df_corr.columns[i]
            cols_a_eliminar.add(colname)

print(f"Se eliminarán {len(cols_a_eliminar)} columnas por alta correlación (> {umbral})")

X_limpia_3 = X_stack_completo.drop(columns=cols_a_eliminar)
X_test_limpia_3 = X_test_stack_completo.drop(columns=cols_a_eliminar)

cols_emb_final = [c for c in X_limpia_3.columns if c in df_embeddings.columns]
cols_meta_final = [c for c in X_limpia_3.columns if c not in df_embeddings.columns]

lr_emb = LogisticRegression(max_iter=1000, random_state=42)
lr_emb.fit(X_limpia_3[cols_emb_final], y)

rf_meta = RandomForestClassifier(n_estimators=200, random_state=42)
rf_meta.fit(X_limpia_3[cols_meta_final], y)


lr_train_probs = lr_emb.predict_proba(X_limpia_3[cols_emb_final])[:, 1]
lr_test_probs = lr_emb.predict_proba(X_test_limpia_3[cols_emb_final])[:, 1]
rf_train_probs = rf_meta.predict_proba(X_limpia_3[cols_meta_final])[:, 1]
rf_test_probs = rf_meta.predict_proba(X_test_limpia_3[cols_meta_final])[:, 1]
stack_train = np.column_stack([lr_train_probs, rf_train_probs])
stack_test = np.column_stack([lr_test_probs, rf_test_probs])

meta_model = LogisticRegression(max_iter=1000, random_state=42)
meta_model.fit(stack_train, y)

final_test_preds = meta_model.predict(stack_test)

# Reporte final
y_test = y_test.astype(int)
print(classification_report(y_test, final_test_preds))

cm = confusion_matrix(y_test, final_test_preds)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
Se eliminarán 10 columnas por alta correlación (> 0.6)
              precision    recall  f1-score   support

           0       0.86      0.81      0.83       139
           1       0.84      0.88      0.86       155

    accuracy                           0.85       294
   macro avg       0.85      0.84      0.85       294
weighted avg       0.85      0.85      0.85       294

<Figure size 800x600 with 0 Axes>
```



XGBoost

```
xgb_model = xgb.XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)

xgb_model.fit(X_escalado, y)
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:200: UserWarning: [18:00:41] WARNING: /__w/xgboost/xgboost/src/l
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

XGBClassifier

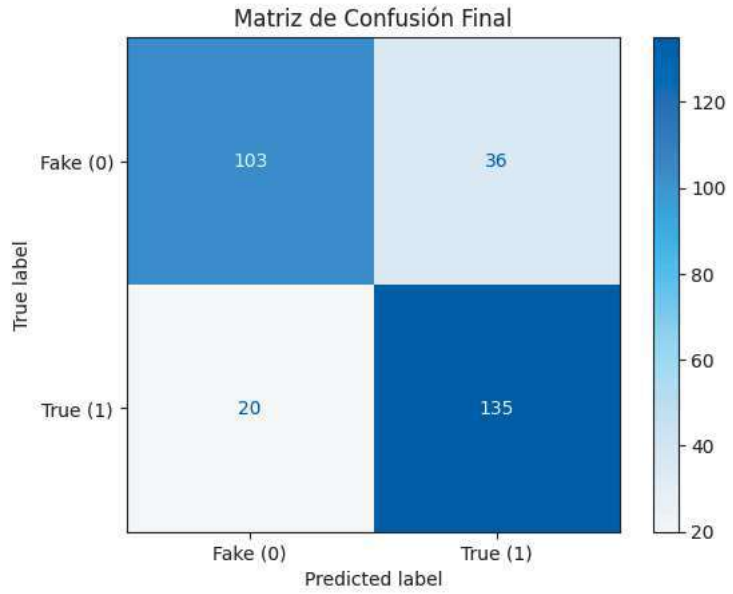
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=6, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=500, n_jobs=None,
```

```
y_pred_test = xgb_model.predict(X_test_escalado)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.84      0.74      0.79       139
           1       0.79      0.87      0.83       155

    accuracy                           0.81       294
   macro avg       0.81      0.81      0.81       294
weighted avg       0.81      0.81      0.81       294
```

```
cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



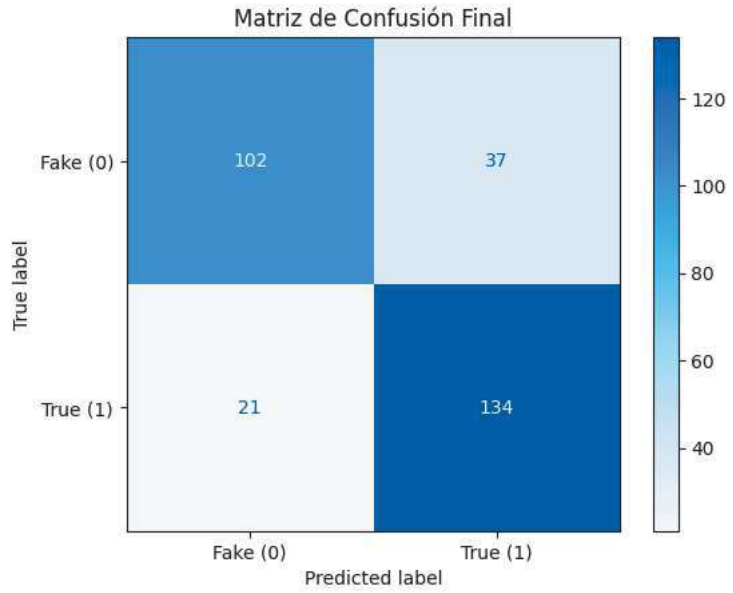

```
#Vuelvo a hacer el XGB pero con la X sin altas correlaciones
xgb_model = xgb.XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)

xgb_model.fit(X_limpia, y)
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:200: UserWarning: [18:00:54] WARNING: /__w/xgboost/xgboost/src/l
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max cat threshold=None. max cat to onehot=None.
```

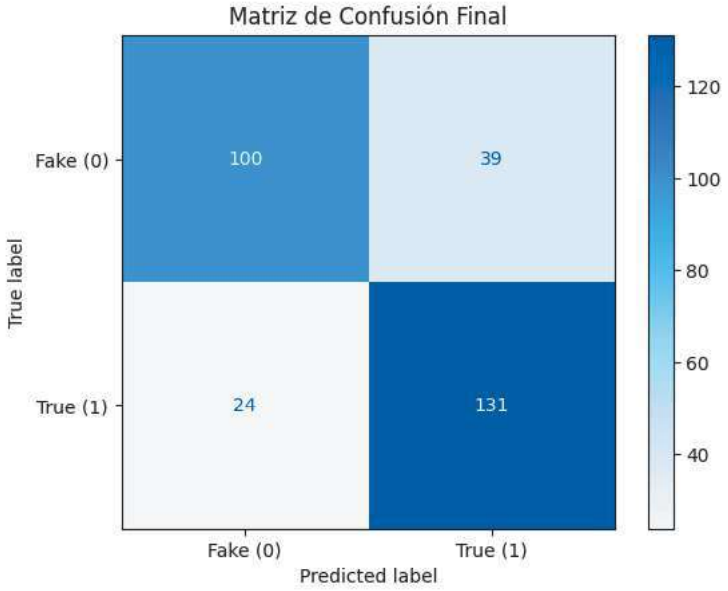
```
y_pred_test = xgb_model.predict(X_test_escalado_limpia)
print(classification_report(y_test, y_pred_test))

cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.83      0.73      0.78       139
           1       0.78      0.86      0.82       155

    accuracy                           0.80       294
   macro avg       0.81      0.80      0.80       294
weighted avg       0.81      0.80      0.80       294

<Figure size 800x600 with 0 Axes>
```




```
#Vuelvo a hacer el XGB pero con las 35 "best features"
xgb_model = xgb.XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)

xgb_model.fit(X_top35, y)

y_pred_test = xgb_model.predict(X_top35_test)
print(classification_report(y_test, y_pred_test))

cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:200: UserWarning: [18:01:07] WARNING: /__w/xgboost/xgboost/src/l
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
              precision    recall  f1-score   support

           0       0.81      0.72      0.76       139
           1       0.77      0.85      0.81       155

    accuracy                           0.79       294
   macro avg       0.79      0.78      0.78       294
weighted avg       0.79      0.79      0.78       294

<Figure size 800x600 with 0 Axes>
```




```
#Probamos el XGBoost pero habiendo hecho PCA sobre los embeddings
xgb_model = xgb.XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)

xgb_model.fit(X_pca, y)
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:200: UserWarning: [18:01:08] WARNING: /__w/xgboost/xgboost/src/l
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

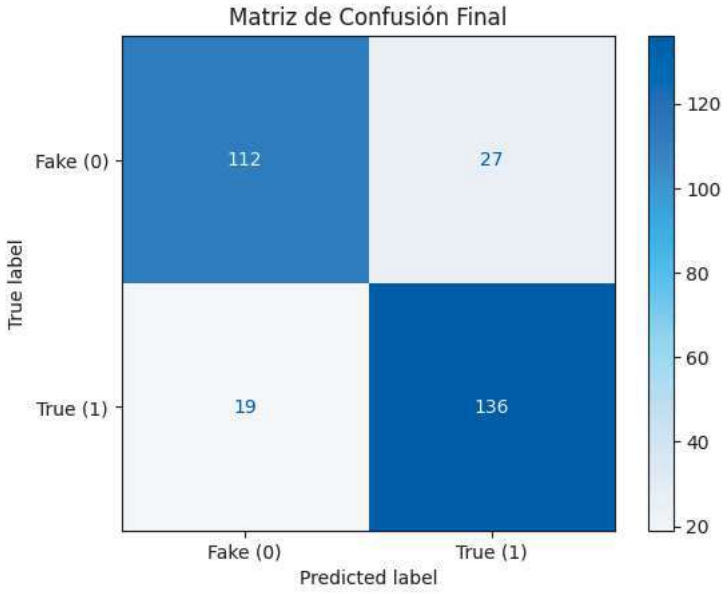
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=6, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=500, n_jobs=None,
```

```
y_pred_test = xgb_model.predict(X_test_pca)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.85      0.81      0.83       139
           1       0.83      0.88      0.86       155

    accuracy                           0.84       294
   macro avg       0.84      0.84      0.84       294
weighted avg       0.84      0.84      0.84       294
```

```
cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

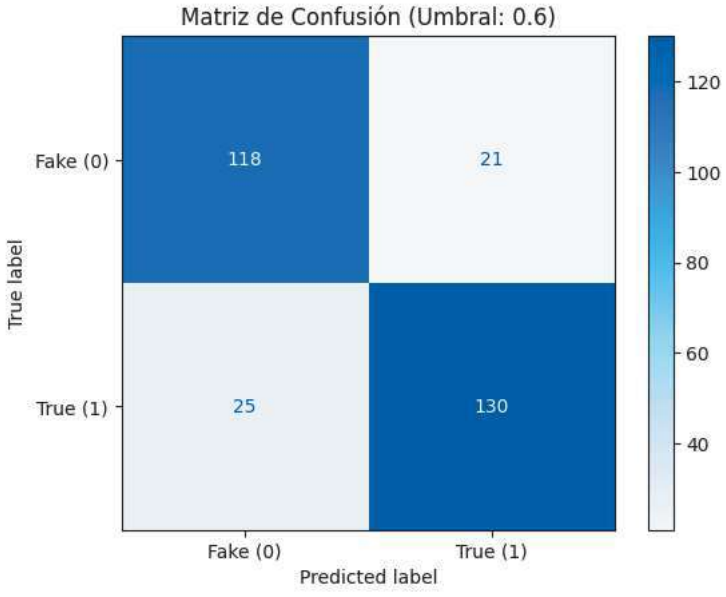



```
#Vuelvo a hacer el XGB pero con la X sin altas correlaciones y con PCA
xgb_model = xgb.XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)

xgb_model.fit(X_limpia_pca, y)

y_pred_test = xgb_model.predict(X_test_limpia_pca)
print(classification_report(y_test, y_pred_test))

cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:200: UserWarning: [18:01:12] WARNING: /__w/xgboost/xgboost/src/l
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
              precision    recall  f1-score   support

           0       0.86      0.79      0.82       139
           1       0.83      0.88      0.85       155

    accuracy                           0.84       294
   macro avg       0.84      0.84      0.84       294
weighted avg       0.84      0.84      0.84       294

<Figure size 800x600 with 0 Axes>
```


```
#Como el mejor modelo es el de PCA sin quitar correlaciones, vamos a probar a subirle el umbral
xgb_model = xgb.XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.1,
    subsamble=0.8,
    colsample_bytree=0.8,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)
xgb_model.fit(X_pca, y)

y_probs = xgb_model.predict_proba(X_test_pca)[:, 1]

umbral_personalizado = 0.6
y_pred_custom = (y_probs >= umbral_personalizado).astype(int)

#Evaluación y Visualización
print(classification_report(y_test, y_pred_custom))
cm_custom = confusion_matrix(y_test, y_pred_custom)

# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm_custom, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues', values_format='d')
plt.title(f'Matriz de Confusión (Umbral: {umbral_personalizado})')
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:200: UserWarning: [18:01:15] WARNING: /__w/xgboost/xgboost/src/l
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
              precision    recall  f1-score   support

           0       0.83      0.85      0.84       139
           1       0.86      0.84      0.85       155

    accuracy                           0.84       294
   macro avg       0.84      0.84      0.84       294
weighted avg       0.84      0.84      0.84       294

<Figure size 800x600 with 0 Axes>
```



SVM

```
svm_model = SVC(
    kernel='rbf',
    C=1.0,
    random_state=42,
    probability=True
)

#Entrenamiento
svm_model.fit(X_escalado, y)
```

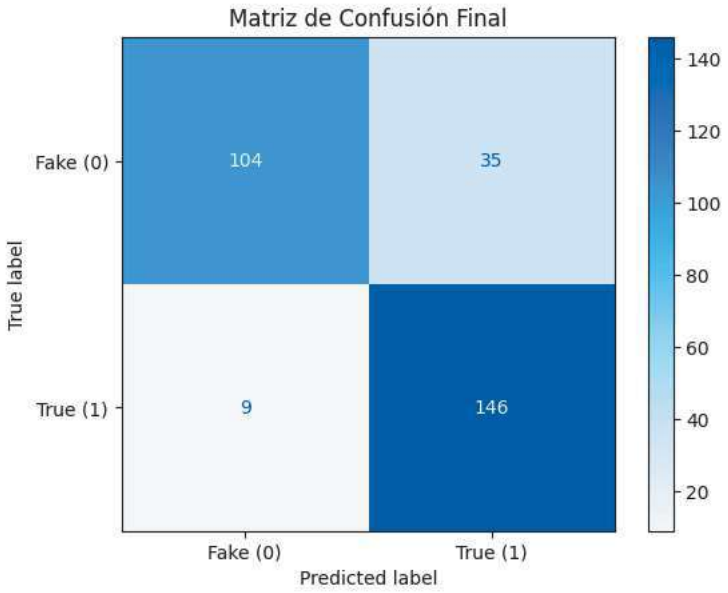
```
SVC(probability=True, random_state=42)
```

```
y_pred_test = svm_model.predict(X_test_escalado)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.92      0.75      0.83       139
           1       0.81      0.94      0.87       155

    accuracy                           0.85       294
   macro avg       0.86      0.85      0.85       294
weighted avg       0.86      0.85      0.85       294
```

```
cm = confusion_matrix(y_test, y_pred_test)
#La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

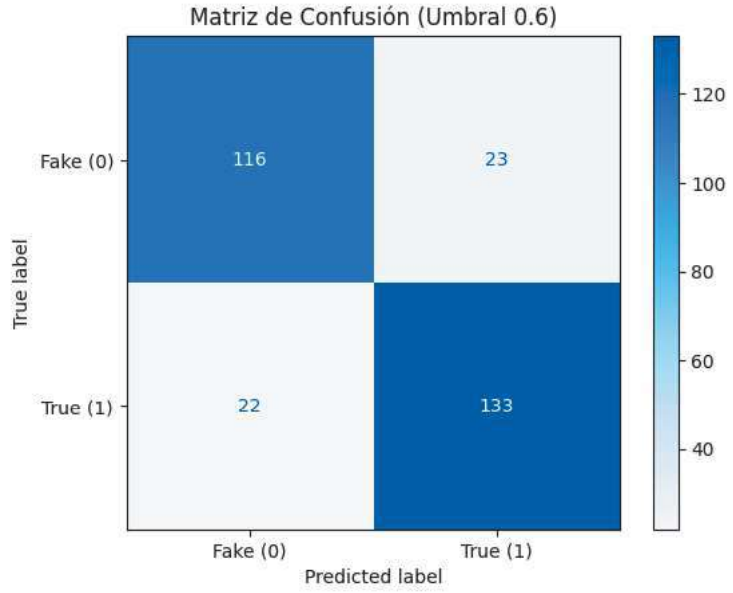
```
<Figure size 800x600 with 0 Axes>
```




```
#Probamos un modelo subiendo el umbral
svm_model = SVC(
    kernel='rbf',
    C=1.0,
    random_state=42,
    probability=True
)

svm_model.fit(X_escalado, y)

probs = svm_model.predict_proba(X_escalado)[:, 1]

#Ajustar el umbral (Threshold)
umbral = 0.6
predicciones_ajustadas = (probs >= umbral).astype(int)
```

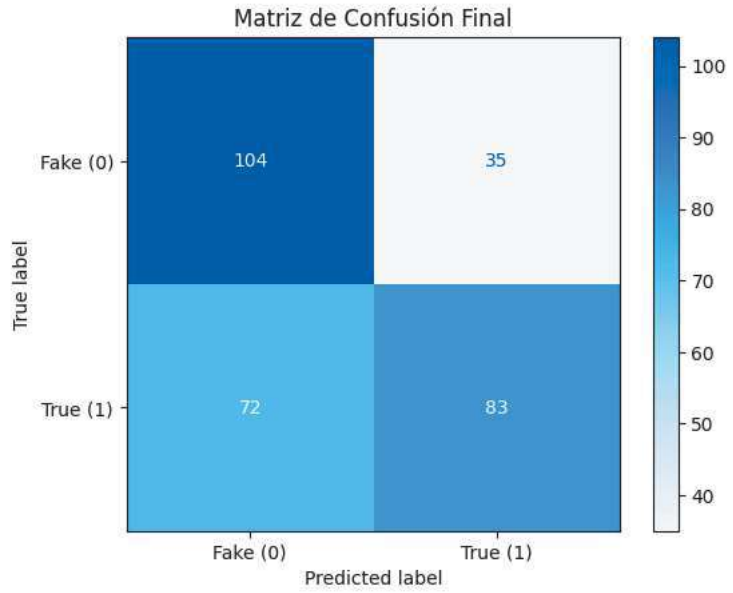
```
probs_test = svm_model.predict_proba(X_test_escalado)[:, 1]

umbral = 0.6
y_pred_ajustada = (probs_test >= umbral).astype(int)

print(classification_report(y_test, y_pred_ajustada))

cm = confusion_matrix(y_test, y_pred_ajustada)

plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title(f'Matriz de Confusión (Umbral {umbral})')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.84      0.83      0.84       139
           1       0.85      0.86      0.86       155

    accuracy                           0.85       294
   macro avg       0.85      0.85      0.85       294
weighted avg       0.85      0.85      0.85       294

<Figure size 800x600 with 0 Axes>
```



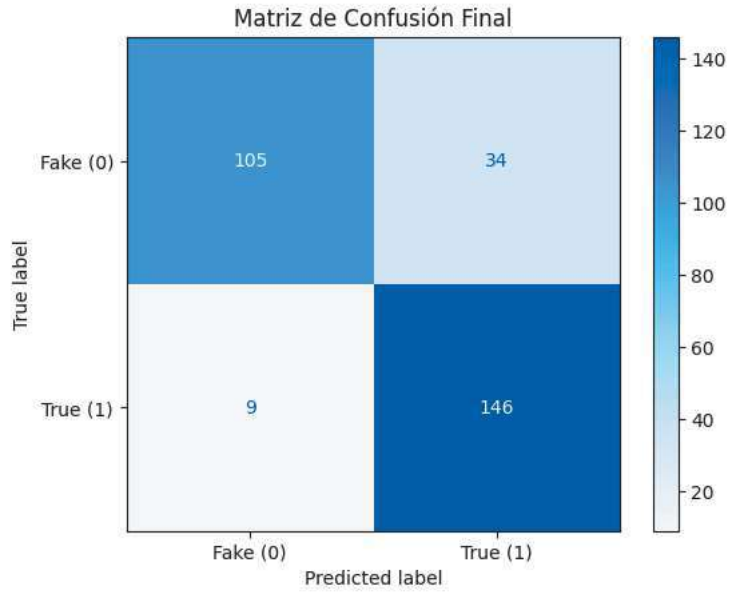

```
#Ahora probamos con PCA
svm_model = SVC(
    kernel='rbf',
    C=1.0,
    random_state=42,
    probability=True
)

#Entrenamiento
svm_model.fit(X_pca, y)

y_pred_test = svm_model.predict(X_test_pca)
print(classification_report(y_test, y_pred_test))

cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.59      0.75      0.66       139
           1       0.70      0.54      0.61       155

    accuracy                           0.64       294
   macro avg       0.65      0.64      0.63       294
weighted avg       0.65      0.64      0.63       294

<Figure size 800x600 with 0 Axes>
```



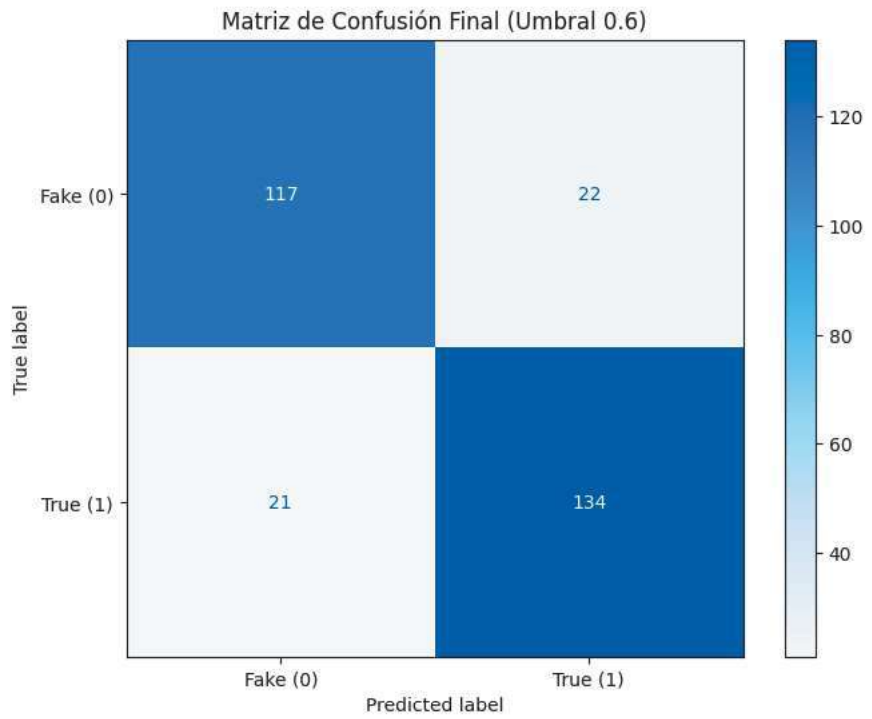

```
#Como empeora con PCA, ahora probamos a quitar las correlaciones sobre el modelo sin PCA
svm_model = SVC(
    kernel='rbf',
    C=1.0,
    random_state=42,
    probability=True
)

#Entrenamiento
svm_model.fit(X_limpia, y)

y_pred_test = svm_model.predict(X_test_escalado_limpia)
print(classification_report(y_test, y_pred_test))

cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.92      0.76      0.83       139
           1       0.81      0.94      0.87       155

    accuracy                           0.85       294
   macro avg       0.87      0.85      0.85       294
weighted avg       0.86      0.85      0.85       294

<Figure size 800x600 with 0 Axes>
```



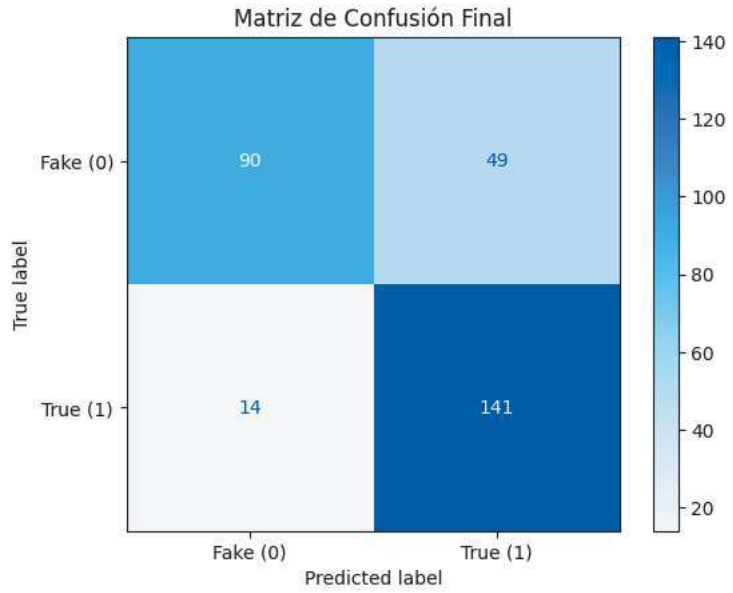

```
#Pruebo a subirle el umbral
svm_model = SVC(
    kernel='rbf',
    C=1.0,
    random_state=42,
    probability=True
)

#Entrenamiento
svm_model.fit(X_limpia, y)

probs_test = svm_model.predict_proba(X_test_escalado_limpia)[:, 1]

umbral = 0.6
y_pred_ajustada = (probs_test >= umbral).astype(int)

#Evaluamos
print(classification_report(y_test, y_pred_ajustada))

#Matriz de confusión
cm = confusion_matrix(y_test, y_pred_ajustada)

plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues', ax=plt.gca())
plt.title(f'Matriz de Confusión Final (Umbral {umbral})')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.85      0.84      0.84       139
           1       0.86      0.86      0.86       155

    accuracy                           0.85       294
   macro avg       0.85      0.85      0.85       294
weighted avg       0.85      0.85      0.85       294
```




```
#Probamos seleccionando las 35 mejores features
svm_model = SVC(
    kernel='rbf',
    C=1.0,
    random_state=42,
    probability=True
)

#Entrenamiento
svm_model.fit(X_top35, y)

y_pred_test = svm_model.predict(X_top35_test)
print(classification_report(y_test, y_pred_test))

cm = confusion_matrix(y_test, y_pred_test)
# La dibujamos
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Fake (0)', 'True (1)'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión Final')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.87      0.65      0.74       139
           1       0.74      0.91      0.82       155

    accuracy                           0.79       294
   macro avg       0.80      0.78      0.78       294
weighted avg       0.80      0.79      0.78       294

<Figure size 800x600 with 0 Axes>
```

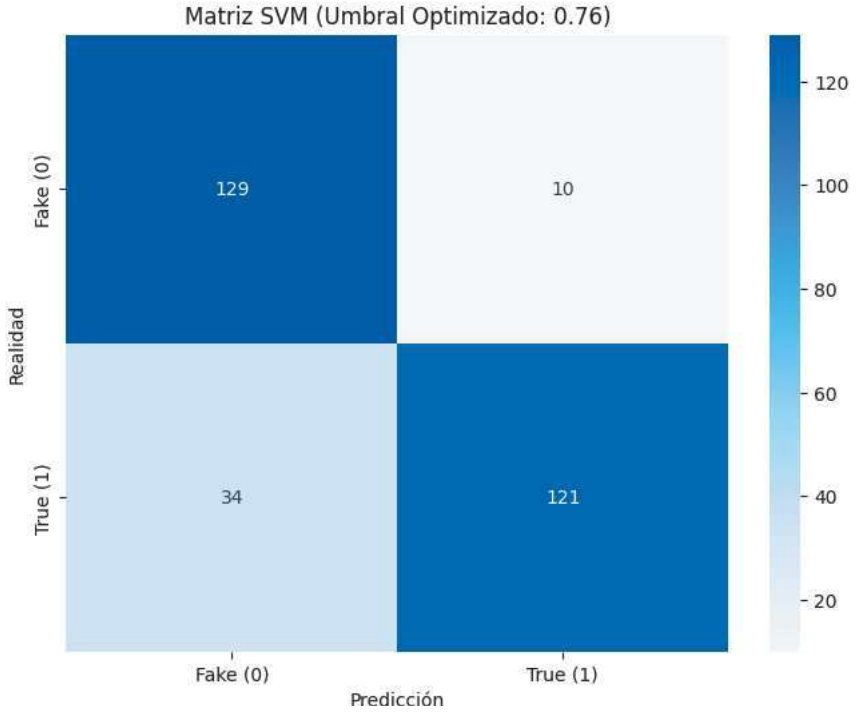


Selección mejor modelo

Probamos a optimizar el umbral de clasificación de los 4 modelos seleccionados como "mejores" para finalmente quedarnos con el mejor de ellos para la detección de noticias falsas

```
# ========================================
# MODELO SVM
# ========================================
svm_model = SVC(
    kernel='rbf',
    C=1.0,
    random_state=42,
    probability=True
)

svm_model.fit(X_limpia, y)

# ========================================
# OBTENCIÓN DE PROBABILIDADES
# ========================================
y_probs_svm = svm_model.predict_proba(X_test_escalado_limpia)[:, 1]

# ========================================
# UMBRAL OPTIMO
# ========================================
mejor_t_svm = 0.5
mejor_prec_svm = 0

for t in np.linspace(0.5, 0.85, 36):
    preds_t = (y_probs_svm >= t).astype(int)

    prec = precision_score(y_test, preds_t, zero_division=0)
    acc = accuracy_score(y_test, preds_t)

    if prec > mejor_prec_svm and acc > 0.80:
        mejor_prec_svm = prec
        mejor_t_svm = t

# ========================================
# APLICACIÓN Y RESULTADOS FINALES
# ========================================
final_preds_svm = (y_probs_svm >= mejor_t_svm).astype(int)

print(f"UMBRAL ÓPTIMO ENCONTRADO PARA SVM: {mejor_t_svm:.2f}")
print(f"Precisión alcanzada: {mejor_prec_svm:.4f}")
print("-" * 45)
print(classification_report(y_test.astype(int), final_preds_svm))

cm_svm = confusion_matrix(y_test.astype(int), final_preds_svm)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Fake (0)', 'True (1)'],
            yticklabels=['Fake (0)', 'True (1)'])
plt.title(f'Matriz SVM (Umbral Optimizado: {mejor_t_svm:.2f})')
plt.xlabel('Predicción')
plt.ylabel('Realidad')
plt.show()
```

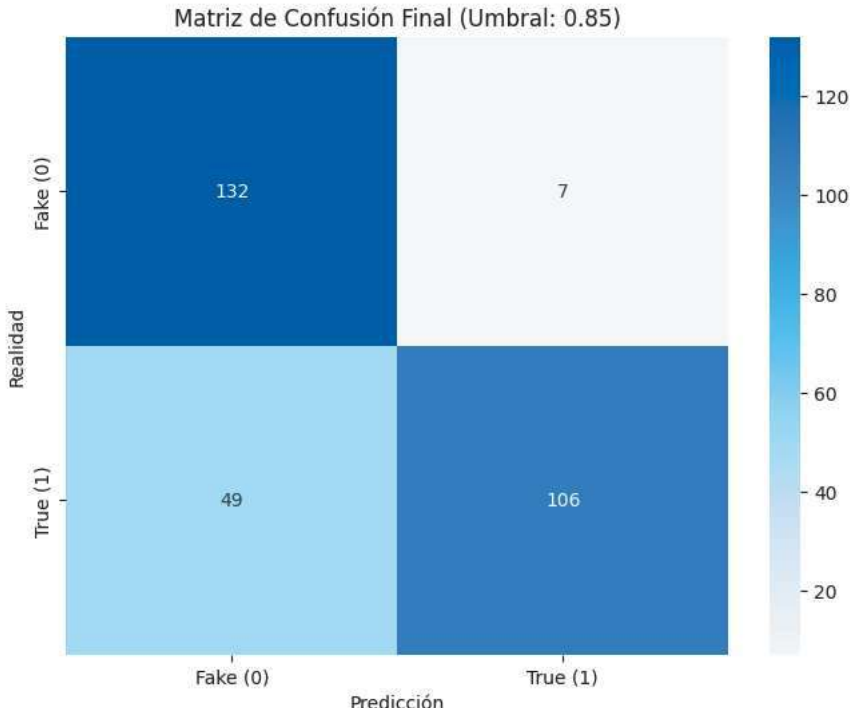
```
UMBRAL ÓPTIMO ENCONTRADO PARA SVM: 0.76
Precisión alcanzada: 0.9237
-----------------------------------------------
              precision    recall  f1-score   support

           0       0.79      0.93      0.85       139
           1       0.92      0.78      0.85       155

    accuracy                           0.85       294
   macro avg       0.86      0.85      0.85       294
weighted avg       0.86      0.85      0.85       294
```



```
# ========================================
# LOGISTIC REGRESSION + RANDOM FOREST
# ========================================
scaler = StandardScaler()
emb_train_scaled = scaler.fit_transform(emb_train_pca)
emb_test_scaled = scaler.transform(emb_test_pca)

df_emb_train = pd.DataFrame(emb_train_scaled, columns=emb_cols_pca, index=X.index)
df_emb_test = pd.DataFrame(emb_test_scaled, columns=emb_cols_pca)

# ========================================
# PREPARACIÓN DE DATOS
# ========================================
X_stack_completo = pd.concat([df_emb_train.reset_index(drop=True), X_no_emb.reset_index(drop=True)], axis=1)
X_test_stack_completo = pd.concat([df_emb_test.reset_index(drop=True), X_test_no_emb.reset_index(drop=True)], axis=1)

umbral_corr = 0.6
df_corr = X_stack_completo.corr().abs()
upper = df_corr.where(np.triu(np.ones(df_corr.shape), k=1).astype(bool))
cols_a_eliminar = [column for column in upper.columns if any(upper[column] > umbral_corr)]

X_limpia = X_stack_completo.drop(columns=cols_a_eliminar)
X_test_limpia = X_test_stack_completo.drop(columns=cols_a_eliminar)

print(f"Columnas eliminadas por correlación (> {umbral_corr}): {len(cols_a_eliminar)}")

cols_emb_final = [c for c in X_limpia.columns if c in emb_cols_pca]
cols_meta_final = [c for c in X_limpia.columns if c not in emb_cols_pca]

# ========================================
# ENTRENAMIENTO
# ========================================
lr_emb = LogisticRegression(max_iter=1000, random_state=42)
lr_emb.fit(X_limpia[cols_emb_final], y)

rf_meta = RandomForestClassifier(n_estimators=200, random_state=42)
rf_meta.fit(X_limpia[cols_meta_final], y)

lr_train_p = lr_emb.predict_proba(X_limpia[cols_emb_final])[:, 1]
rf_train_p = rf_meta.predict_proba(X_limpia[cols_meta_final])[:, 1]
lr_test_p = lr_emb.predict_proba(X_test_limpia[cols_emb_final])[:, 1]
rf_test_p = rf_meta.predict_proba(X_test_limpia[cols_meta_final])[:, 1]

stack_train = np.column_stack([lr_train_p, rf_train_p])
stack_test = np.column_stack([lr_test_p, rf_test_p])

meta_model = LogisticRegression(max_iter=1000, random_state=42)
meta_model.fit(stack_train, y)

# ========================================
# OPTIMIZACIÓN DEL UMBRAL
# ========================================
probs_finales = meta_model.predict_proba(stack_test)[:, 1]

mejor_t = 0.5
mejor_prec = 0
for t in np.linspace(0.5, 0.85, 36):
    preds_t = (probs_finales >= t).astype(int)
    prec = precision_score(y_test, preds_t, zero_division=0)
    acc = accuracy_score(y_test, preds_t)

    if prec > mejor_prec and acc > 0.80:
        mejor_prec = prec
        mejor_t = t

final_preds_opt = (probs_finales >= mejor_t).astype(int)

# ========================================
# VISUALIZACIÓN Y REPORTE
# ========================================
print(f"\n UMBRAL OPTIMIZADO: {mejor_t:.2f}")
print("-" * 40)
print(classification_report(y_test.astype(int), final_preds_opt))

cm = confusion_matrix(y_test.astype(int), final_preds_opt)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Fake (0)', 'True (1)'],
            yticklabels=['Fake (0)', 'True (1)'])
plt.title(f'Matriz de Confusión Final (Umbral: {mejor_t:.2f})')
plt.xlabel('Predicción')
```

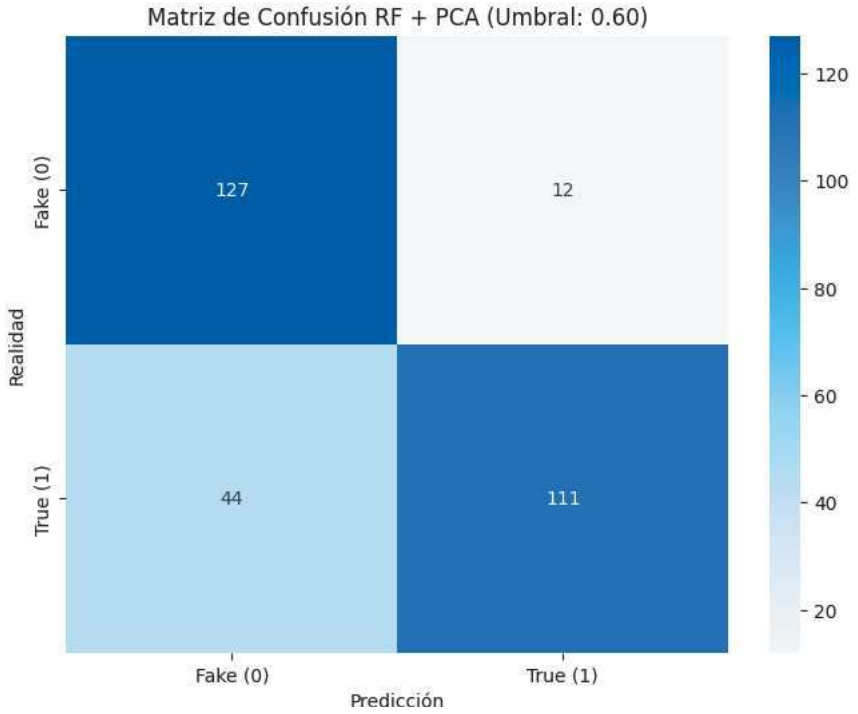
```
plt.ylabel('Realidad')
plt.show()
```

```
Columnas eliminadas por correlación (> 0.6): 10

 UMBRAL OPTIMIZADO: 0.85
---------------------------------------
              precision    recall  f1-score   support

           0       0.73      0.95      0.82       139
           1       0.94      0.68      0.79       155

    accuracy                           0.81       294
   macro avg       0.83      0.82      0.81       294
weighted avg       0.84      0.81      0.81       294
```



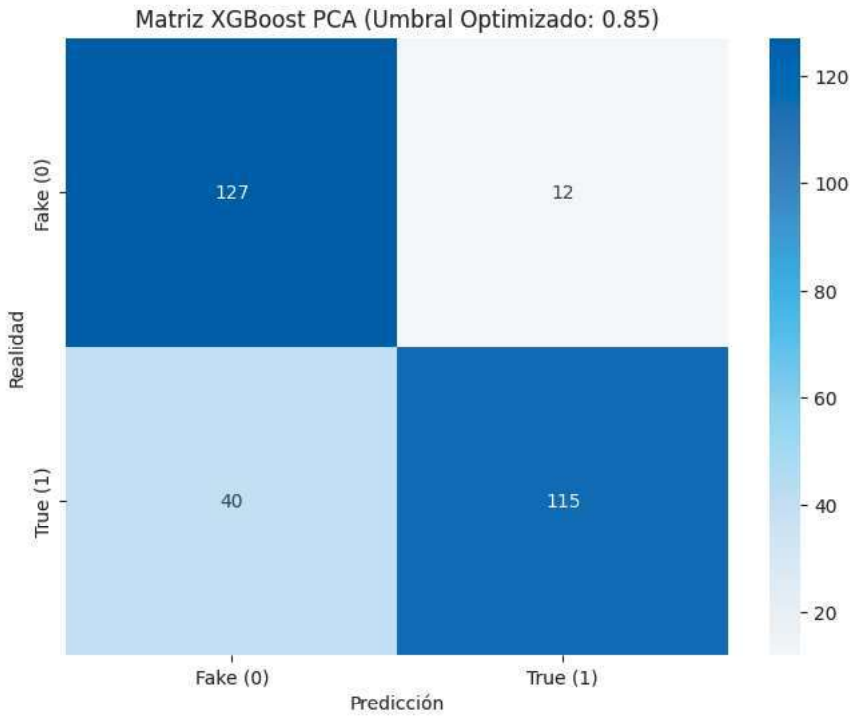
```
# ========================================
# RANDOM FOREST
# ========================================
umbral = 0.6
df_corr = X_pca.corr().abs()
upper = df_corr.where(np.triu(np.ones(df_corr.shape), k=1).astype(bool))
cols_a_eliminar = [column for column in upper.columns if any(upper[column] > umbral)]

print(f"Se eliminarán {len(cols_a_eliminar)} columnas por alta correlación (> {umbral})")

X_limpia_pca = X_pca.drop(columns=cols_a_eliminar)
X_test_limpia_pca = X_test_pca.drop(columns=cols_a_eliminar)

# ========================================
# ENTRENAMIENTO
# ========================================
rf_model = RandomForestClassifier(n_estimators=200, random_state=42)
rf_model.fit(X_limpia_pca, y)

# ========================================
# OPTIMIZACIÓN DEL UMBRAL
# ========================================
probs_test = rf_model.predict_proba(X_test_limpia_pca)[:, 1]

mejor_t = 0.5
mejor_prec = 0

for t in np.linspace(0.5, 0.85, 36):
    preds_t = (probs_test >= t).astype(int)
    prec = precision_score(y_test.astype(int), preds_t, zero_division=0)
    acc = accuracy_score(y_test.astype(int), preds_t)

    if prec > mejor_prec and acc > 0.80:
        mejor_prec = prec
        mejor_t = t

final_preds = (probs_test >= mejor_t).astype(int)

# ========================================
# REPORTES Y MATRIZ DE CONFUSIÓN
# ========================================
print(f"\n UMBRAL OPTIMIZADO: {mejor_t:.2f}")
print("-" * 40)
print(classification_report(y_test.astype(int), final_preds))

# Dibujar Matriz
cm = confusion_matrix(y_test.astype(int), final_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Fake (0)', 'True (1)'],
            yticklabels=['Fake (0)', 'True (1)'])
plt.title(f'Matriz de Confusión RF + PCA (Umbral: {mejor_t:.2f})')
plt.xlabel('Predicción')
plt.ylabel('Realidad')
plt.show()
```

```
Se eliminarán 10 columnas por alta correlación (> 0.6)

 UMBRAL OPTIMIZADO: 0.60
----------------------------------------
              precision    recall  f1-score   support

           0       0.74      0.91      0.82       139
           1       0.90      0.72      0.80       155

    accuracy                           0.81       294
   macro avg       0.82      0.81      0.81       294
weighted avg       0.83      0.81      0.81       294
```



```
# ========================================
# XGBOOST
# ========================================
xgb_model_pca = xgb.XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)

xgb_model_pca.fit(X_pca, y)

# ========================================
# PROBABILIDADES
# ========================================
y_probs_xgb = xgb_model_pca.predict_proba(X_test_pca)[:, 1]

# ========================================
# UMBRAL OPTIMO
# ========================================
mejor_t = 0.5
mejor_prec = 0

for t in np.linspace(0.5, 0.85, 36):
    preds_t = (y_probs_xgb >= t).astype(int)

    prec = precision_score(y_test, preds_t, zero_division=0)
    acc = accuracy_score(y_test, preds_t)

    if prec > mejor_prec and acc > 0.80:
        mejor_prec = prec
        mejor_t = t

# ========================================
# APLICACIÓN Y RESULTADOS FINALES
# ========================================
final_preds_xgb = (y_probs_xgb >= mejor_t).astype(int)

print(f" UMBRAL ÓPTIMO ENCONTRADO PARA XGBOOST: {mejor_t:.2f}")
print(f"Precisión alcanzada: {mejor_prec:.4f}")
print("-" * 45)
print(classification_report(y_test.astype(int), final_preds_xgb))

cm_xgb = confusion_matrix(y_test.astype(int), final_preds_xgb)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Fake (0)', 'True (1)'],
            yticklabels=['Fake (0)', 'True (1)'])
plt.title(f'Matriz XGBoost PCA (Umbral Optimizado: {mejor_t:.2f})')
plt.xlabel('Predicción')
plt.ylabel('Realidad')
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:200: UserWarning: [18:01:39] WARNING: /__w/xgboost/xgboost/src/l
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
 UMBRAL ÓPTIMO ENCONTRADO PARA XGBOOST: 0.85
Precisión alcanzada: 0.9055
-----------------------------------------------
              precision    recall  f1-score   support

           0       0.76      0.91      0.83       139
           1       0.91      0.74      0.82       155

    accuracy                           0.82       294
   macro avg       0.83      0.83      0.82       294
weighted avg       0.84      0.82      0.82       294
```

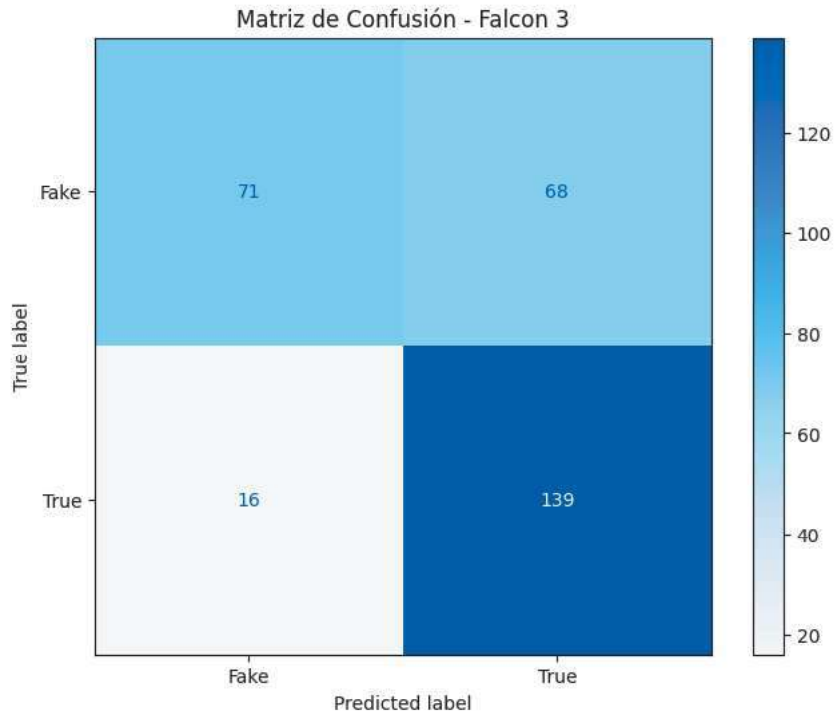


Analisis de LLM

```
df = df_backup
MODEL_ID = "tiiuae/Falcon3-1B-Instruct"
MAX_LENGTH = 256

def limpiar_df(df_input):
    """Asegura que las etiquetas sean exactas antes de procesar."""
    df_input = df_input.copy()
    df_input['Category'] = df_input['Category'].astype(str).str.strip().str.capitalize()
    df_input = df_input[df_input['Category'].isin(['True', 'Fake'])]
    return df_input

df = limpiar_df(df)
development = limpiar_df(development)
test = limpiar_df(test_backup)

def preparar_dataset(df_input):
    def format_prompt(row):
        return f"### Noticia: {row['Headline']}\n### Contenido: {str(row['Text'])[:400]}\n### Clasificación: {row['Category
    temp_df = df_input.copy()
    temp_df["text"] = temp_df.apply(format_prompt, axis=1)
    return Dataset.from_pandas(temp_df[["text"]])

train_ds = preparar_dataset(df)
dev_ds = preparar_dataset(development)


#Cargamos el modelo
tokenizer = AutoTokenizer.from_pretrained(MODEL_ID, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True,
)

model = AutoModelForCausalLM.from_pretrained(
    MODEL_ID,
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True,
    low_cpu_mem_usage=True
)

model = prepare_model_for_kbit_training(model)

lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules="all-linear",
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)
model = get_peft_model(model, lora_config)


#Entrenamiento
def tokenize_func(examples):
    result = tokenizer(examples["text"], truncation=True, max_length=MAX_LENGTH, padding="max_length")
    result["labels"] = result["input_ids"].copy()
    return result

tokenized_train = train_ds.map(tokenize_func, batched=True)
tokenized_dev = dev_ds.map(tokenize_func, batched=True)

training_args = TrainingArguments(
    output_dir="./fake-news-falcon",
    per_device_train_batch_size=4,
    gradient_accumulation_steps=4,
    num_train_epochs=3,
    learning_rate=2e-4,
    weight_decay=0.01,
    logging_steps=10,
    fp16=True,
    eval_strategy="steps",
    eval_steps=50,
    save_strategy="no",
    report_to="none"
)
```

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_dev,
    data_collator=DataCollatorForLanguageModeling(tokenizer, mlm=False)
)

trainer.train()

#Evaluación
def limpiar_salida_llm(texto):
    texto = texto.lower().strip()
    if texto.startswith("f"): return "Fake"
    if texto.startswith("t"): return "True"
    return "Desconocido"

def evaluar_modelo(model, tokenizer, test_df):
    test_ds = preparar_dataset(test_df)
    y_true, y_pred = [], []

    model.eval()
    device = next(model.parameters()).device

    print(f"Generando predicciones sobre {len(test_ds)} ejemplos...")

    for i in range(len(test_ds)):
        full_text = test_ds[i]['text']
        prompt = full_text.split("Clasificación:")[0] + "Clasificación:"
        etiqueta_real = full_text.split("Clasificación:")[1].replace("<|endoftext|>", "").strip()

        inputs = tokenizer(prompt, return_tensors="pt").to(device)

        with torch.no_grad():
            outputs = model.generate(
                **inputs,
                max_new_tokens=2,
                pad_token_id=tokenizer.eos_token_id,
                do_sample=False
            )

        respuesta_cruda = tokenizer.decode(outputs[0][inputs['input_ids'].shape[1]:], skip_special_tokens=True)

        y_true.append(etiqueta_real)
        y_pred.append(limpiar_salida_llm(respuesta_cruda))

    target_names = ["Fake", "True"]
    print("\n" + "="*30)
    print("INFORME DE CLASIFICACIÓN")
    print("="*30)
    print(classification_report(y_true, y_pred, labels=target_names))

    #matriz de confusión
    cm = confusion_matrix(y_true, y_pred, labels=target_names)
    fig, ax = plt.subplots(figsize=(8, 6))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=target_names)
    disp.plot(cmap=plt.cm.Blues, ax=ax)
    plt.title("Matriz de Confusión - Falcon 3")
    plt.show()

#Test
evaluar_modelo(model, tokenizer, test)
```

```
Loading weights: 100%                                   165/165 [00:13<00:00, 20.13it/s]

Map: 100%                                               1174/1174 [00:01<00:00, 1088.17 examples/s]

Map: 100%                                               295/295 [00:00<00:00, 838.06 examples/s]
/usr/local/lib/python3.12/dist-packages/torch/_dynamo/eval_frame.py:1181: UserWarning: torch.utils.checkpoint: the use_reent
  return fn(*args, **kwargs)
{'loss': '3.439', 'grad_norm': '0.5094', 'learning_rate': '0.0001919', 'epoch': '0.1361'}
{'loss': '2.972', 'grad_norm': '0.4302', 'learning_rate': '0.0001829', 'epoch': '0.2721'}
{'loss': '2.94', 'grad_norm': '0.4197', 'learning_rate': '0.0001739', 'epoch': '0.4082'}
{'loss': '2.878', 'grad_norm': '0.4886', 'learning_rate': '0.0001649', 'epoch': '0.5442'}
{'loss': '2.919', 'grad_norm': '0.4742', 'learning_rate': '0.0001559', 'epoch': '0.6803'}
{'eval_loss': '2.679', 'eval_runtime': '22.44', 'eval_samples_per_second': '13.15', 'eval_steps_per_second': '1.649', 'epoch
{'loss': '2.806', 'grad_norm': '0.4859', 'learning_rate': '0.0001468', 'epoch': '0.8163'}
{'loss': '2.803', 'grad_norm': '0.4851', 'learning_rate': '0.0001378', 'epoch': '0.9524'}
{'loss': '2.767', 'grad_norm': '0.4861', 'learning_rate': '0.0001288', 'epoch': '1.082'}
{'loss': '2.718', 'grad_norm': '0.5781', 'learning_rate': '0.0001198', 'epoch': '1.218'}
{'loss': '2.718', 'grad_norm': '0.582', 'learning_rate': '0.0001108', 'epoch': '1.354'}
{'eval_loss': '2.549', 'eval_runtime': '22.42', 'eval_samples_per_second': '13.16', 'eval_steps_per_second': '1.65', 'epoch'
{'loss': '2.7', 'grad_norm': '0.6363', 'learning_rate': '0.0001018', 'epoch': '1.49'}
{'loss': '2.648', 'grad_norm': '0.7258', 'learning_rate': '9.279e-05', 'epoch': '1.626'}
{'loss': '2.676', 'grad_norm': '0.7016', 'learning_rate': '8.378e-05', 'epoch': '1.762'}
{'loss': '2.635', 'grad_norm': '0.7373', 'learning_rate': '7.477e-05', 'epoch': '1.898'}
{'loss': '2.688', 'grad_norm': '0.7594', 'learning_rate': '6.577e-05', 'epoch': '2.027'}
{'eval_loss': '2.451', 'eval_runtime': '22.48', 'eval_samples_per_second': '13.12', 'eval_steps_per_second': '1.646', 'epoch
{'loss': '2.503', 'grad_norm': '0.737', 'learning_rate': '5.676e-05', 'epoch': '2.163'}
{'loss': '2.527', 'grad_norm': '0.8051', 'learning_rate': '4.775e-05', 'epoch': '2.299'}
{'loss': '2.47', 'grad_norm': '0.8065', 'learning_rate': '3.874e-05', 'epoch': '2.435'}
{'loss': '2.628', 'grad_norm': '0.8321', 'learning_rate': '2.973e-05', 'epoch': '2.571'}
{'loss': '2.574', 'grad_norm': '0.8436', 'learning_rate': '2.072e-05', 'epoch': '2.707'}
{'eval_loss': '2.396', 'eval_runtime': '22.44', 'eval_samples_per_second': '13.15', 'eval_steps_per_second': '1.649', 'epoch
{'loss': '2.526', 'grad_norm': '0.8541', 'learning_rate': '1.171e-05', 'epoch': '2.844'}
{'loss': '2.548', 'grad_norm': '0.8422', 'learning_rate': '2.703e-06', 'epoch': '2.98'}
{'eval_loss': '2.388', 'eval_runtime': '22.47', 'eval_samples_per_second': '13.13', 'eval_steps_per_second': '1.647', 'epoch
{'train_runtime': '1076', 'train_samples_per_second': '3.272', 'train_steps_per_second': '0.206', 'train_loss': '2.729', 'ep
Generando predicciones sobre 294 ejemplos...

==============================
INFORME DE CLASIFICACIÓN
==============================
              precision    recall  f1-score   support

        Fake       0.82      0.51      0.63       139
        True       0.67      0.90      0.77       155

    accuracy                           0.71       294
   macro avg       0.74      0.70      0.70       294
weighted avg       0.74      0.71      0.70       294
```

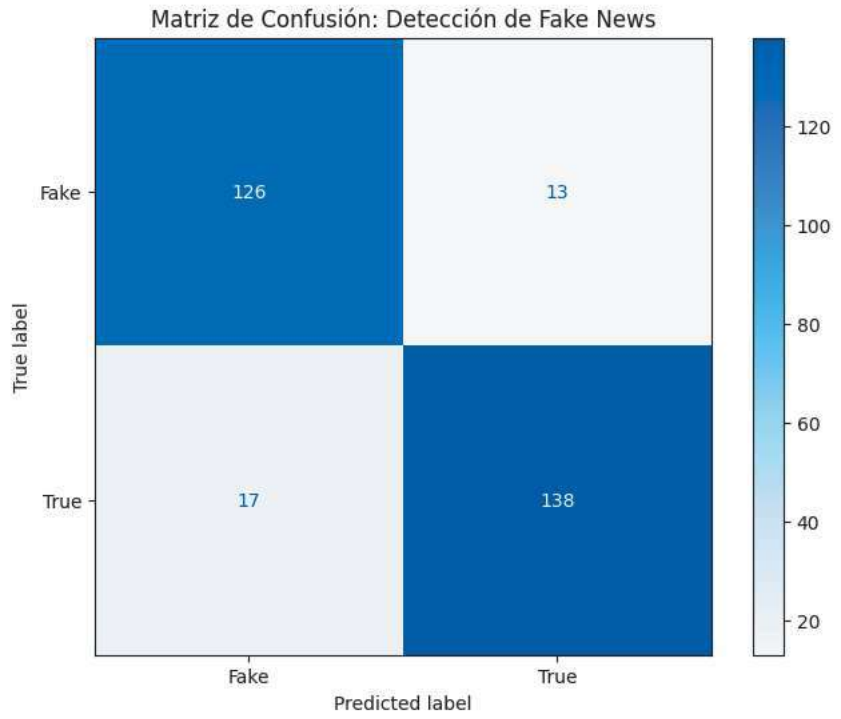


Aqui solo ha evaluado la noticia no ha evaluado tambien la fuente que hemos visto que añadia mucha información sobre si es true o false. Ahora voy a probar a añadirla para la clasificación

```
#Fijo valores
def fix_randomness(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    os.environ['PYTHONHASHSEED'] = str(seed)
    print(f"Determinismo fijado con la semilla: {seed}")

fix_randomness(42)

#Configuración
MODEL_ID = "tiiuae/Falcon3-1B-Instruct"
MAX_LENGTH = 512

tokenizer = AutoTokenizer.from_pretrained(MODEL_ID, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"

#Tokenización
def tokenize_with_masking(examples):
    all_input_ids = []
    all_labels = []

    for i in range(len(examples["Source"])):
        prompt_info = (f"Instrucción: Clasifica como 'True' o 'Fake'.\n"
                       f"Fuente: {examples['Source'][i]}\n"
                       f"Titular: {examples['Headline'][i]}\n"
                       f"Contenido: {str(examples['Text'][i])[:300]}\n"
                       f"Respuesta:")

        full_text = f"{prompt_info} {examples['Category'][i]}<|endoftext|>"

        prompt_tokens = tokenizer(prompt_info, truncation=True, max_length=MAX_LENGTH)
        full_tokens = tokenizer(full_text, truncation=True, max_length=MAX_LENGTH, padding="max_length")

        ids = full_tokens["input_ids"]
        labels = [-100] * MAX_LENGTH

        start_idx = len(prompt_tokens["input_ids"])

        for j in range(start_idx, MAX_LENGTH):
            if ids[j] == tokenizer.pad_token_id:
                break
            labels[j] = ids[j]

        all_input_ids.append(ids)
        all_labels.append(labels)

    return {"input_ids": all_input_ids, "labels": all_labels}

ds_train = Dataset.from_pandas(df)
tokenized_train = ds_train.map(tokenize_with_masking, batched=True, remove_columns=ds_train.column_names)

#Modelo
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16
)

model = AutoModelForCausalLM.from_pretrained(
    MODEL_ID, quantization_config=bnb_config, device_map="auto", trust_remote_code=True
)

model = prepare_model_for_kbit_training(model)
model = get_peft_model(model, LoraConfig(
    r=16, lora_alpha=32, target_modules="all-linear", lora_dropout=0.05, task_type="CAUSAL_LM"
))

#Entrenamiento
training_args = TrainingArguments(
    output_dir="./fake-news-final",
    per_device_train_batch_size=4,
    gradient_accumulation_steps=4,
    num_train_epochs=3,
    learning_rate=1e-4,
    fp16=True,
    logging_steps=10,
    save_strategy="no",
```

```
    remove_unused_columns=False,
    full_determinism=True,
    seed=42,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
)

trainer.train()

#Evaluación
def evaluar_modelo(model, tokenizer, df_eval):
    y_true, y_pred = [], []
    model.eval()

    print("Iniciando evaluación...")
    for _, row in df_eval.iterrows():
        prompt = (f"Instrucción: Clasifica como 'True' o 'Fake'.\n"
                  f"Fuente: {row['Source']}\n"
                  f"Titular: {row['Headline']}\n"
                  f"Contenido: {str(row['Text'])[:300]}\n"
                  f"Respuesta:")

        inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

        with torch.no_grad():
            outputs = model.generate(
                **inputs,
                max_new_tokens=3,
                do_sample=False,
                pad_token_id=tokenizer.eos_token_id
            )

        raw_output = tokenizer.decode(outputs[0][inputs["input_ids"].shape[1]:], skip_special_tokens=True).strip().lower()

        if "fake" in raw_output: pred = "Fake"
        elif "true" in raw_output: pred = "True"
        else: pred = "Desconocido"

        y_true.append(row['Category'])
        y_pred.append(pred)

    print("\n--- Reporte de Clasificación ---")
    print(classification_report(y_true, y_pred))
    return y_true, y_pred

# Matriz de confusión
y_true_final, y_pred_final = evaluar_modelo(model, tokenizer, test_backup)

cm = confusion_matrix(y_true_final, y_pred_final, labels=["Fake", "True"])
fig, ax = plt.subplots(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Fake", "True"])

disp.plot(cmap="Blues", ax=ax, values_format='d')
plt.title("Matriz de Confusión: Detección de Fake News")
plt.show()
```

```
Determinismo fijado con la semilla: 42
Map: 100%                                                    1174/1174 [00:02<00:00, 538.92 examples/s]
Loading weights: 100%                                        165/165 [00:13<00:00, 20.34it/s]
/usr/local/lib/python3.12/dist-packages/torch/_dynamo/eval_frame.py:1181: UserWarning: torch.utils.checkpoint: the use_reent
  return fn(*args, **kwargs)
{'loss': '1.669', 'grad_norm': '4.671', 'learning_rate': '9.595e-05', 'epoch': '0.1361'}
{'loss': '0.7913', 'grad_norm': '2.017', 'learning_rate': '9.144e-05', 'epoch': '0.2721'}
{'loss': '0.7001', 'grad_norm': '6.014', 'learning_rate': '8.694e-05', 'epoch': '0.4082'}
{'loss': '0.655', 'grad_norm': '1.954', 'learning_rate': '8.243e-05', 'epoch': '0.5442'}
{'loss': '0.5591', 'grad_norm': '1.837', 'learning_rate': '7.793e-05', 'epoch': '0.6803'}
{'loss': '0.5236', 'grad_norm': '5.459', 'learning_rate': '7.342e-05', 'epoch': '0.8163'}
{'loss': '0.5537', 'grad_norm': '7.316', 'learning_rate': '6.892e-05', 'epoch': '0.9524'}
{'loss': '0.4126', 'grad_norm': '2.005', 'learning_rate': '6.441e-05', 'epoch': '1.082'}
{'loss': '0.3367', 'grad_norm': '7.561', 'learning_rate': '5.991e-05', 'epoch': '1.218'}
{'loss': '0.4082', 'grad_norm': '8.297', 'learning_rate': '5.541e-05', 'epoch': '1.354'}
{'loss': '0.2686', 'grad_norm': '1.877', 'learning_rate': '5.09e-05', 'epoch': '1.49'}
{'loss': '0.1912', 'grad_norm': '4.487', 'learning_rate': '4.64e-05', 'epoch': '1.626'}
{'loss': '0.1989', 'grad_norm': '2.567', 'learning_rate': '4.189e-05', 'epoch': '1.762'}
{'loss': '0.1782', 'grad_norm': '2.258', 'learning_rate': '3.739e-05', 'epoch': '1.898'}
{'loss': '0.2168', 'grad_norm': '0.3672', 'learning_rate': '3.288e-05', 'epoch': '2.027'}
{'loss': '0.1407', 'grad_norm': '6.135', 'learning_rate': '2.838e-05', 'epoch': '2.163'}
{'loss': '0.1027', 'grad_norm': '4.617', 'learning_rate': '2.387e-05', 'epoch': '2.299'}
{'loss': '0.1193', 'grad_norm': '1.595', 'learning_rate': '1.937e-05', 'epoch': '2.435'}
{'loss': '0.07276', 'grad_norm': '0.1913', 'learning_rate': '1.486e-05', 'epoch': '2.571'}
{'loss': '0.1813', 'grad_norm': '2.753', 'learning_rate': '1.036e-05', 'epoch': '2.707'}
{'loss': '0.1187', 'grad_norm': '5.101', 'learning_rate': '5.856e-06', 'epoch': '2.844'}
{'loss': '0.1646', 'grad_norm': '8.296', 'learning_rate': '1.351e-06', 'epoch': '2.98'}
{'train_runtime': '1836', 'train_samples_per_second': '1.919', 'train_steps_per_second': '0.121', 'train_loss': '0.3864', 'e
Iniciando evaluación...

--- Reporte de Clasificación ---
              precision    recall  f1-score   support

        Fake       0.88      0.91      0.89       139
        True       0.91      0.89      0.90       155

    accuracy                           0.90       294
   macro avg       0.90      0.90      0.90       294
weighted avg       0.90      0.90      0.90       294
```

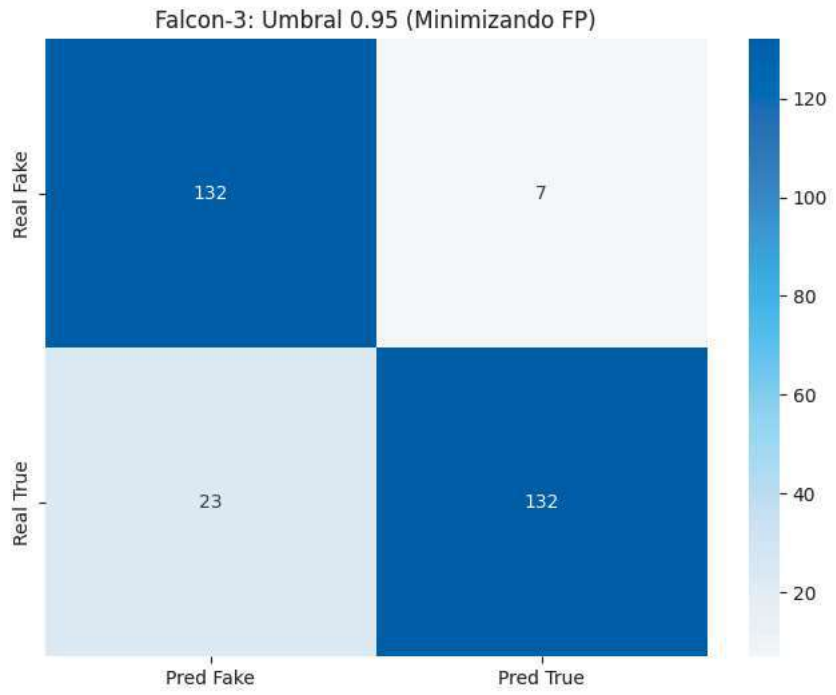


En el primer modelo el LLM es entrenado para poder predecir todo el texto lo cual lo hace ineficiente. En el segundo modelo aprende exclusivamente a clasidicar True o Fake pero ya en este tercer modelo, mejor en vez de que prediga True o Fake lo que hago es que prediga una probabilidad y si la probabiliad es superior al umbral es True y sino es Fake. Al establecerlo como una probabilidad se puede trabajar más facilmente la optimización del umbral de clasificación.

```
#fIjo valores
def fix_randomness(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    os.environ['PYTHONHASHSEED'] = str(seed)

fix_randomness(42)

#Inicio el modelo
MODEL_ID = "tiiuae/Falcon3-1B-Instruct"
MAX_LENGTH = 512

tokenizer = AutoTokenizer.from_pretrained(MODEL_ID, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"

#Entrenamiento
def tokenize_with_masking(examples):
    all_input_ids, all_labels = [], []
    for i in range(len(examples["Source"])):
        prompt_info = (f"Instrucción: Clasifica como 'True' o 'Fake'.\n"
                       f"Fuente: {examples['Source'][i]}\n"
                       f"Titular: {examples['Headline'][i]}\n"
                       f"Contenido: {str(examples['Text'][i])[:300]}\n"
                       f"Respuesta:")
        full_text = f"{prompt_info} {examples['Category'][i]}<|endoftext|>"
        prompt_tokens = tokenizer(prompt_info, truncation=True, max_length=MAX_LENGTH)
        full_tokens = tokenizer(full_text, truncation=True, max_length=MAX_LENGTH, padding="max_length")
        ids = full_tokens["input_ids"]
        labels = [-100] * MAX_LENGTH
        start_idx = len(prompt_tokens["input_ids"])
        for j in range(start_idx, MAX_LENGTH):
            if ids[j] == tokenizer.pad_token_id: break
            labels[j] = ids[j]
        all_input_ids.append(ids); all_labels.append(labels)
    return {"input_ids": all_input_ids, "labels": all_labels}

ds_train = Dataset.from_pandas(df)
tokenized_train = ds_train.map(tokenize_with_masking, batched=True, remove_columns=ds_train.column_names)

bnb_config = BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_quant_type="nf4", bnb_4bit_compute_dtype=torch.float16)
model = AutoModelForCausalLM.from_pretrained(MODEL_ID, quantization_config=bnb_config, device_map="auto", trust_remote_code
model = prepare_model_for_kbit_training(model)
model = get_peft_model(model, LoraConfig(r=16, lora_alpha=32, target_modules="all-linear", lora_dropout=0.05, task_type="CA

trainer = Trainer(
    model=model,
    train_dataset=tokenized_train,
    args=TrainingArguments(output_dir="./fake-news-falcon", per_device_train_batch_size=4, gradient_accumulation_steps=4,
                           num_train_epochs=3, learning_rate=1e-4, fp16=True, save_strategy="no", full_determinism=True, se
)
trainer.train()

#Probabilidades
def obtener_probabilidades_llm(model, tokenizer, df_eval):
    model.eval()
    id_true = tokenizer.encode("True", add_special_tokens=False)[-1]
    id_fake = tokenizer.encode("Fake", add_special_tokens=False)[-1]

    lista_confianza_true = []
    y_reales = []

    for _, row in tqdm(df_eval.iterrows(), total=len(df_eval)):
        prompt = (f"Instrucción: Clasifica como 'True' o 'Fake'.\n"
                  f"Fuente: {row['Source']}\n"
                  f"Titular: {row['Headline']}\n"
                  f"Contenido: {str(row['Text'])[:300]}\n"
                  f"Respuesta:")
        inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
        with torch.no_grad():
            outputs = model(**inputs)
            logits = outputs.logits[:, -1, :]
            probs = torch.softmax(logits, dim=-1)
            p_true = probs[0, id_true].item()
            p_fake = probs[0, id_fake].item()
            confianza = p_true / (p_true + p_fake) if (p_true + p_fake) > 0 else 0
            lista_confianza_true.append(confianza)
            y_reales.append(row['Category'])
```

```
    return np.array(lista_confianza_true), np.array(y_reales)

# Umbral
confianzas, y_reales = obtener_probabilidades_llm(model, tokenizer, test_backup)
y_reales_bin = np.where(y_reales == "True", 1, 0)
mejor_umbral = 0.5
mejor_precision = 0
min_accuracy = 0.80

for t in np.linspace(0.4, 0.95, 56):
    preds_t = (confianzas >= t).astype(int)
    prec = precision_score(y_reales_bin, preds_t, zero_division=0)
    acc = accuracy_score(y_reales_bin, preds_t)

    if prec > mejor_precision and acc >= min_accuracy:
        mejor_precision = prec
        mejor_umbral = t

#Resultados finales
preds_finales = np.where(confianzas >= mejor_umbral, "True", "Fake")

print(f"\n Umbral Óptimo Seleccionado: {mejor_umbral:.3f}")
print(f"Precisión (Evitar FP): {mejor_precision:.3f}")
print("-" * 40)
print(classification_report(y_reales, preds_finales))

cm = confusion_matrix(y_reales, preds_finales, labels=["Fake", "True"])
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Pred Fake', 'Pred True'], yticklabels=['Real Fake', 'Real
plt.title(f"Falcon-3: Umbral {mejor_umbral:.2f} (Minimizando FP)")
plt.show()
```

```
Map: 100%                                                    1174/1174 [00:02<00:00, 535.20 examples/s]

Loading weights: 100%                                                    165/165 [00:13<00:00, 21.74it/s]

/usr/local/lib/python3.12/dist-packages/torch/_dynamo/eval_frame.py:1181: UserWarning: torch.utils.checkpoint: the use_reent
  return fn(*args, **kwargs)
{'train_runtime': '1838', 'train_samples_per_second': '1.916', 'train_steps_per_second': '0.121', 'train_loss': '0.3864', 'e
100%|██████████| 294/294 [00:47<00:00,  6.13it/s]

 Umbral Óptimo Seleccionado: 0.950
Precisión (Evitar FP): 0.950
----------------------------------------
              precision    recall  f1-score   support

        Fake       0.85      0.95      0.90       139
        True       0.95      0.85      0.90       155

    accuracy                           0.90       294
   macro avg       0.90      0.90      0.90       294
weighted avg       0.90      0.90      0.90       294
```



```
#Miramos a ver en que ha fallado exactamente
def analizar_errores(df_eval, y_true, y_pred, confianzas, umbral):
    errores = []
    for i in range(len(y_true)):
        if y_true[i] != y_pred[i]:
            errores.append({
                'Headline': df_eval.iloc[i]['Headline'],
                'Source': df_eval.iloc[i]['Source'],
                'Real': y_true[i],
                'Predicho': y_pred[i],
                'Confianza_Score': round(confianzas[i], 4),
                'Distancia_Umbral': round(abs(confianzas[i] - umbral), 4)
            })
    return pd.DataFrame(errores)

df_errores = analizar_errores(test_backup, y_reales, preds_finales, confianzas, mejor_umbral)

# Ordenamos los errores de mayor a menor segun la probabilidad que habia asignado el modelo
if not df_errores.empty:
    df_errores = df_errores.sort_values(by='Distancia_Umbral', ascending=False)

print(f"\n Total de errores encontrados: {len(df_errores)}")
print(df_errores.head(30))
```

```
 Total de errores encontrados: 30
                                             Headline  \
24  Los termómetros infrarrojos no matan las neuro...
19  El Gobierno no sabrá el nombre de quien se man...
9   NO PHONE: EL "DISPOSITIVO" QUE NO HACE NADA PA...
25  Los disquetes de 3.5 pulgadas se niegan a mori...
14  Irreverente. Zopilotes no se suicidan cuando e...
2   Jorge Sonnante: «Hay personas que están empeza...
15  Alerta en la República Democrática del Congo p...
20  lo que NBC no mostró de la entrevista con Putin\n
16  ¿Aviones militares para repatriar a los clande...
12  Se casó con su tía abuela de *NUMBER* años, en...
5                    Regresa La Academia, de Tv Azteca
29  ERUVIEL ÁVILA DEJA EL PRI CAPITALINO PARA UNIR...
7   Julen Lopetegui es el nuevo DT del Real Madrid...
23  Como nunca, Peña Nieto felicita a su hijo Dieg...
6   Tres secretarios federales han muerto en accid...
4   Elena Poniatowska: 'AMLO será un presidente mu...
28  Academia de policia ignacio zaragoza tendra ce...
10  Campaña de desprestigio y amenazas contra Carm...
3   México: Aumenta la violencia en las protestas ...
22  Montero agita el 8-M con charlas sobre insurge...
21  En dos hospitales privados el covid-19 es 100%...
13                     El coronavirus llega a Cúcuta
26                Hospitales en La Laguna, al límite
18  Científicos: La barba contiene más bacterias d...
8   Una veintena de abertzales patean a un guardia...
0   Arzobispo primado de México respalda uniones c...
27  100.000 venezolanos se sumaron al censo electo...
11  “No tengamos miedo, vacunémonos”, dice Magdale...
1   COVID-19 en España, hoy | Sanidad notifica 590...
17  Terrible: ¡¡Sánchez subvencionó con dinero púb...

                    Source  Real Predicho  Confianza_Score  Distancia_Umbral
24           Salud con Lupa  True     Fake           0.0226            0.9274
19           NoticiaTrabajo  True     Fake           0.0675            0.8825
9                  Merca2.0  True     Fake           0.0920            0.8580
25                 Xataka  True     Fake           0.1176            0.8324
14             SDP Noticia  True     Fake           0.1520            0.7980
2   Noticia & protagonistas  True     Fake           0.2878            0.6622
15                El Clarin  True     Fake           0.3040            0.6460
20              sputnkinews  True     Fake           0.3124            0.6376
16                   Bladi  True     Fake           0.3329            0.6171
12                El Clarin  True     Fake           0.4017            0.5483
5                Dos Mundos  True     Fake           0.4785            0.4715
29                Expansión  True     Fake           0.4805            0.4695
7              Medio Tiempo  True     Fake           0.4863            0.4637
23                   Quien  True     Fake           0.4961            0.4539
6                 Expansión  True     Fake           0.4980            0.4520
4                 Nacion321  True     Fake           0.5794            0.3706
28       Tribuna Noticias   True     Fake           0.7432            0.2068
10       Aristegui noticias  True     Fake           0.8634            0.0866
3                 France 24  True     Fake           0.8856            0.0644
22                 La Razon  True     Fake           0.8948            0.0552
21                  Milenio  Fake     True           0.9998            0.0498
13            La República  Fake     True           0.9992            0.0492
26            El Universal  Fake     True           0.9991            0.0491
```

```
#Ejemplo de como funcionaría un modelo LLM para la detección de noticias falsas

def clasificar_noticia_manual(titular, contenido, fuente="Desconocida"):
    model.eval()

    id_true = tokenizer.encode(" True", add_special_tokens=False)[-1]
    id_fake = tokenizer.encode(" Fake", add_special_tokens=False)[-1]

    prompt = (f"Instrucción: Clasifica como 'True' o 'Fake'.\n"
              f"Fuente: {fuente}\n"
              f"Titular: {titular}\n"
              f"Contenido: {str(contenido)[:300]}\n"
              f"Respuesta:")

    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits[:, -1, :]
        probs = torch.softmax(logits, dim=-1)

        p_true = probs[0, id_true].item()
        p_fake = probs[0, id_fake].item()

        suma_total = p_true + p_fake
        confianza_true = p_true / suma_total if suma_total > 0 else 0

    resultado = "TRUE" if confianza_true >= mejor_umbral else "FAKE"

    #Resultados
    print(f"\n[Análisis de Falcon-3 Fine-tuned]")
    print(f"----------------------------------")
    print(f"Prob. True: {p_true:.6f} | Prob. Fake: {p_fake:.6f}")
    print(f"Confianza Final (Relativa): {confianza_true:.4f}")
    print(f"Umbral aplicado: {mejor_umbral:.4f}")
    print(f"Veredicto: {resultado}")
    print(f"----------------------------------")

    return resultado

#Prueba 1
titular_test = "Nueva York es la capital de Estados Unidos"
cuerpo_test = "Nueva York es la capital de unos de los paises del conteniente de America"
fuente= "El País"

clasificar_noticia_manual(titular_test, cuerpo_test, fuente)

#Prueba 2
titular_test = "El Papa se cambia el nombre a Papa Trump"
cuerpo_test = "El Papa ha decidido que en honor a Donald Trump y a sus intentos de establecer la paz se va a cambiar el nor
fuente= "Laguna"

clasificar_noticia_manual(titular_test, cuerpo_test, fuente)
```

```
[Análisis de Falcon-3 Fine-tuned]
----------------------------------
Prob. True: 0.999932 | Prob. Fake: 0.000011
Confianza Final (Relativa): 1.0000
Umbral aplicado: 0.9500
Veredicto: TRUE
----------------------------------

[Análisis de Falcon-3 Fine-tuned]
----------------------------------
Prob. True: 0.080276 | Prob. Fake: 0.918712
Confianza Final (Relativa): 0.0804
Umbral aplicado: 0.9500
Veredicto: FAKE
----------------------------------
'FAKE'
```