ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

IDENTIFICACIÓN AUTOMÁTICA DE NOTICIAS FALSAS EN ESPAÑOL UTILIZANDO TÉCNICAS DE MINERÍA DE DATOS Y PROCESAMIENTO DE LENGUAJE NATURAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO MATEMÁTICO

PROYECTO DE INVESTIGACIÓN

NICOLÁS JESÚS MAFLA CHECA
nicolas.mafla.checa@gmail.com

Director: DR. MIGUEL ALFONSO SÁNCHEZ FLORES
miguel.flores@epn.edu.ec

QUITO, ABRIL 2021

DECLARACIÓN

Yo NICOLÁS JESÚS MAFLA CHECA, declaro bajo juramento que el trabajo aquí escrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual, correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normatividad institucional vigente.

Nicolás Jesús Mafla Checa

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por NICOLÁS JESÚS MAFLA CHECA, bajo mi supervisión.

Dr. Miguel Alfonso Sánchez Flores
Director del Proyecto

AGRADECIMIENTOS

A Dios por guiarme en el camino. A mi familia por cuidarme toda la vida. A Miguel por depositar su confianza en mi. A Abigail por apoyarme en cada decisión tomada. Y a mis amigos por sus consejos y lealtad

DEDICATORIA

A todos los que creyeron en mi desde el inicio

Índice general

Índice de figuras

Índice de tablas

Resumen

La circulación de noticias falsas, en especial las de sátira política, por medios de comunicación digitales ha afectado a la mayoría de la población ecuatoriana por su masiva propagación y la falta de regulación, por lo que en este trabajo se presenta una metodología fundamentada en aprendizaje estadístico que detecta de manera precisa y automática noticias falsas basándose en la forma en que están escritas. Para esto se utilizó lenguaje de procesamiento natural, algoritmos clásicos de clasificación supervisada y el lenguaje de programación Python. El documento inicia por definir todos los conceptos necesarios para comprender tanto las noticias falsas como los algoritmos de aprendizaje estadístico, posteriormente se explica como fueron recolectadas las noticias y procesadas, para luego detallar el proceso de entrenamiento de los modelos y finalmente la presentación y análisis de los respectivos resultados.

Palabras clave: Noticias falsas, procesamiento del lenguaje natural, aprendizaje estadístico, español, minería de datos.

Abstract

The circulation of false news, especially political satire type, via digital media has affected the majority of the Ecuadorian population due to its massive spread and lack of regulation, which is why in this paper presents a methodology grounded in statistical learning that accurately and automatically detects fake news based on the way it is written. For this, natural processing language, classical supervised classification algorithms and the Python programming language were used. The document begins by defining all the necessary concepts to understand both false news and statistical learning algorithms, later it is explained how the news was collected and processed, to then detail the process of training the models and finally the presentation and analysis of the respective results.

Keywords: Fake news, natural language processing, statistical learning, spanish, data mining.

Capítulo 1

Introducción

El tiempo que invertimos actualmente navegando por internet y consumiendo contenido en redes sociales, ocupa una gran parte de nuestro día, sin duda alguna se han convertido en una de las herramientas de comunicación más poderosas utilizadas en la actualidad dado que permite el acceso al consumo y divulgación de información de manera instantánea al alcance de cualquier persona, esto ha provocado la preferencia de los medios digitales sobre los medios tradicionales como la radio, televisión y prensa. Consecuencia de este libre acceso y la fácil generación de contenido permite que la información que circula en linea no siempre sea confiable (López-Borrull, Vives y Badell, 2018). El desarrollo de las redes sociales aumenta la difusión de noticias falsas en internet, la información distribuida por estos medios es masiva, rápida y heterogénea, por lo que puede causar graves impacto en toda la sociedad (Zhang y Ghorbani, 2019).

Las noticias falsas han tenido un impacto negativo en varios sectores, entre estos, la comunicación, economía, salud y política los más importantes; mencionando algunos casos tenemos: las elecciones presidenciales de Estados Unidos del año 2016, en donde se originó una inmensa cantidad de información falsa en redes sociales acerca del ex presidente Donald Trump para desprestigiar su campaña política (Allcott y Gentzkow, 2017), el posicionamiento digital de un restaurante inexistente como el mejor de Londres en el año 2017, en una página web turística con comentarios que burlaron el algoritmo de seguridad de la plataforma (Rodríguez-Fernández, 2019), la amenaza a la salud pública mundial provocada por la infodemia masiva que se originó entorno a la pandemia ocasionada por el virus covid-19 ya que puede redujo la eficacia de programas, campañas e iniciativas dirigidas a la salud, la concienciación y el bienestar de las personas (Pulido, Ruiz-Eugenio, Redondo-Sama

y Villarejo-Carballido, 2020), un caso curioso es el de una publicación científica en el American Journal of Biomedical Science & Research, la cual afirmaba que comer carne de cierto Pokémon (criaturas ficticias de una popular franquicia de videojuegos) parecido a un murciélago provocó la propagación del covid-19 en una ciudad ficticia (Shelomi, 2020), haciendo notar que incluso la comunidad científica no está exenta a la filtración de investigaciones con información supuestamente verificada.

El crecimiento exponencial en el volumen de noticias falsas que circulan en la red y su gran viralidad, han despertado el interés en la comunidad científica en desarrollar métodos automatizados para su detección temprana, tanto así, que empresas como Facebook y Google han gastado grandes cantidades de dinero en el desarrollo de soluciones (Zhang y Ghorbani, 2019). Hoy en día el estado del arte respecto a la detección de noticias falasas se encuentra en el entrenamiento tanto de algoritmos clásicos de clasificación supervisada como también de redes neuronales a partir conjuntos de noticias manualmente etiquetadas, estos conjuntos pasan previamente por un proceso extracción de características (Bondielli y Marcelloni, 2019); cabe mencionar que la mayor parte de los progresos que se ha tenido relacionado a la identificación de noticias falsas ha sido en el idioma inglés.

La extracción de información en textos no es una tarea fácil, de hecho, es uno de los principales retos que se han presentado dentro de la inteligencia artificial, y para abordar esta temática se ha optado por el uso de varios enfoques incluyendo las técnicas de procesamiento del lenguaje natural (NLP). No se puede seguir una regla general para enfrentarnos a un problema de este tipo ya que varias de estas técnicas dependen mucho del idioma en el que se está trabajando y esto determinará si la tarea a resolver es más simple o compleja (Baeza-Yates, 2004). Algunos de los aportes que se ha tenido del uso de técnicas de NLP tenemos: análisis de sentimientos (Deng y Wiebe, 2016), clasificación de texto (Howard y Ruder, 2018), corrección gramatical (Clément, Gerdes y Marlet, 2011), sistemas de recomendación (Kapoor, Vishal y K. S., 2020), entre otros.

Dado la diversidad de formas de hablar el español, que dependen en gran medida de la zona geográfica de donde proviene el hablante, incluso dentro de nuestro país, el presente trabajo se enfocará en la creación de un modelo capaz de identificar noticias falsas en español provenientes de fuentes ecuatorianas. El modelo será entrenado a partir de un conjunto de publicaciones identificadas como noticias reales y falsas, extraídas manualmente de los principales periódicos nacionales y páginas de sátira política, además se comparan distintos tipos de algoritmos de clasificación utilizados en la detección de noticias falsas (Bondielli y Marcelloni, 2019).

1.1. Justificación

Durante los últimos años y sobre todo en esta última década, una de las ramas de estudio dentro de la inteligencia artificial que ha jugado un papel protagónico en la transformación digital es el aprendizaje estadístico, esta disciplina se basa en predecir una medición de resultados ya sea cuantitativa o cualitativa basándose en un conjunto de características, estimando una función que los relacione, y así, resolver tareas que para los humanos serían imposibles realizarlas ya sea por el tiempo que tomaría resolverlas o su complejidad (James, et. al, 2013).

Varios son los algoritmos que se usan para resolver problemas de clasificación supervisada, los que más se usa en la actualidad son: Regresión logística, Máquinas de soporte vectorial, Bosques aleatorios, Clasificador de naive Bayes, Boosting y Redes neuronales artificiales; el desempeño de cada uno de los algoritmos depende mucho del conjunto de datos y la problemática, esto determinará la prevalencia de unos algoritmos sobre otros (Mohri, Rostamizadeh y Talwalkar, 2018).

El desarrollo de modelos capaces de identificar noticias falsas desde el enfoque de problemas de clasificación supervisada ha ganado popularidad los últimos años, más aún con los problemas suscitados en las elecciones presidenciales del año 2016 en Estados Unidos (Alcott y Gentzkow, 2017). La mayoría de los modelos creados hasta la actualidad están entrenados sobre bases de datos de noticias en idioma ingles disponibles en diferentes repositorios digitales, estas constan de noticias verdaderas obtenidas de páginas oficiales de medios de comunicación escritos y noticias falsas de páginas maliciosas y de servicios de Fact-Checking según Zhang y Ghorbani (2019) y Bondielli y Marcelloni (2019).

El problema de las noticias falsas que circulan en internet y también en redes sociales nos afecta a todos directa o indirectamente, es por esta razón que esta investigación proporciona las bases teóricas para la creación de aplicaciones que sirvan de ayuda para lucha contra la desinformación mediática; como se mencionó anteriormente son muchos los sectores que se ven afectados. Adicionalmente, el desarrollo que se logre potencialmente servirá para futuras investigaciones sobre esta temática y será un avance respecto a modelos de clasificación supervisada que involucren técnicas de procesamiento de lenguaje natural en el idioma español.

1.2. Objetivos

Crear un modelo que permita identificar de manera precisa y automática noticias falsas en español de páginas de noticias y sátiras ecuatorianas

- Crear un corpus de texto extrayendo los datos manualmente o utilizando técnicas de webscraping
- Procesar el texto con técnicas de NLP para crear la base de datos que servirá para entrenar los algoritmos
- Elaborar un modelo para cada uno de los algoritmos de clasificación seleccionados y comparar sus resultados

Capítulo 2

Identificación de noticias falsas

A mediados de los años noventa, con el desarrollo explosivo de la World Wide Web o comúnmente llamado internet, se revolucionó la forma en que las personas se comunican. Hoy en día las redes sociales, como Twitter y Facebook, pueden facilitar la distribución de información en tiempo real entre los usuarios de cualquier parte del mundo, siendo testigos del gran crecimiento en el número de noticias falsas que circulan en linea, que sin darnos cuenta, ahora forman parte de nuestras vidas y estamos expuestos a este tipo de información maliciosa día tras día. Sin duda alguna hemos podido experimentar por cuenta propia cuan incomodas pueden llegar a ser las noticias falsas, ya que estas tienen un impacto profundo tanto en las personas como en la sociedad (Zhang y Ghorbani, 2020). Por tanto, la importancia de la creación de sistemas de detección eficaces para la identificación de noticias falsas es decisivo en la lucha contra la desinformación.

2.1. Noticias falsas

Las noticias falsas o fake news son recientes y han ganado popularidad los últimos años, pero la información falsa ha sido parte de la humanidad durante mucho tiempo, son tan antiguas como las noticias impresas que circulaban desde la invención de la imprenta, o incluso más (Soll, 2016). No existe una definición universal para noticias falsas y no es fácil formular una generalmente aceptada para el término, ya que estas, tienden a ser diversas en cuestión de temas, estilos e incluso plataformas, por lo que varios autores han propuesto sus propias definiciones. Se define las noticias falsas como información fabricada que imita el contenido de los medios de comunicación en forma pero no en el proceso organizacional o intención,

carecen de las normas y procesos editoriales de los medios de comunicación para asegurar la exactitud y credibilidad de información (Lazer et al., 2018). Se refieren a todo tipo de historias falsas que se publican y distribuyen principalmente en internet, con el fin de engañar, burlar o atraer deliberadamente a los lectores para obtener ganancias financieras, políticas o de otro tipo (Zhang y Ghorbani, 2020). Como última definición se tiene que las noticias falsas son artículos de noticias que son intencionalmente creados y verificablemente falsos que podrían engañar a los lectores (Allcott y Gentzkow, 2017). En base a como fueron creados los conceptos presentados, estos comparten tres características en común, la primera es relacionado a la autenticidad de la información de las noticias (conteniendo alguna declaración basada en hechos o no), la segunda se refiere a la intención con la que son creadas (con el objetivo de engañar o entretener al público), y por último, si realmente son noticias (Zhou y Zafarani, 2020).

2.1.1. Caracterización

Para tener una visión más general sobre la gran variedad de los tipos de información falsa que circulan en internet, se debe entender los cuatro elementos principales que la componen. La caracterización de las noticias falsas está basada en (Zhang y Ghorbani, 2020)

Creador/Difusor

Los creadores y difusores de información falsa dentro de internet pueden ser *personas reales* o *no humanos*. Las noticias falsas creadas por personas reales se incluye tanto autores y usuarios que publican información falsa o errónea de manera no intencionada, y usuarios malintencionados que su objetivo es crearla, lo que hace realmente difícil distinguir entre información falsa e información veraz y debido a su naturaleza anónima, los usuarios de internet no necesitan asumir responsabilidades por lo que publican, comparten y comentan. Por otro lado, los cradores de noticias falsas no humanos más comunes son los *social bots*, los cuales consisten en algoritmos informáticos que están diseñados para exhibir comportamientos similares a los humanos y producir automáticamente contenido e interactuar con usuarios de las redes sociales, y los *cyborgs* que se refieren a personas asistidas por bots o viceversa, de manera similar a los social bots los cyborgs engañan a los usuarios de las redes sociales al difundir información y mensajes falsos, lo que puede dañar la creencia y la confianza social.

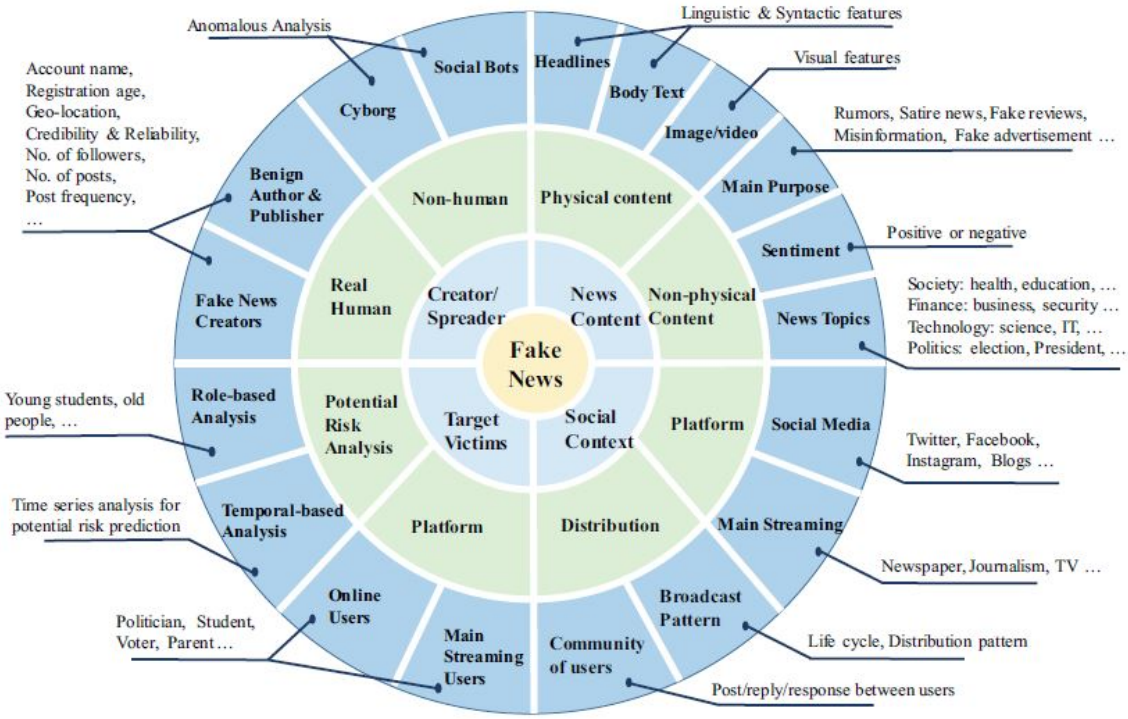

Figura 2.1: Esquema gráfico de los cuatro componentes principales: Creador/Difusor, víctimas objetivo, contenido de noticias y contexto social (Zhang y Ghorbani, 2020, p. 4)

Víctimas objetivo

Los usuarios objetivo o víctimas de las noticias falsas pueden variar dependiendo del propósito utilizado para la creación de esta información maliciosa. Mencionando algunos ejemplos tenemos: los candidatos en época de elecciones pueden ser los usuarios objetivo de reclamos políticos falsos, los clientes en línea pueden ser los usuarios objetivo de reseñas y anuncios falsos en línea, los padres pueden ser los usuarios objetivo de noticias educativas falsas y las personas mayores pueden ser el usuario objetivo de noticias de salud falsas.

Contenido de noticias

El contenido físico consta del título de la noticia, el cuerpo principal de la noticia y los demás elementos como imágenes o videos. Los datos de redes sociales como los tweets de Twitter o las publicaciones de Facebook se convierten en un medio poderoso para compartir información, este contenido proporcionado por los datos en redes sociales son fuentes valiosas y significativas de noticias. En cambio el contenido no físico se refiere a las opiniones, emociones, actitudes y sentimientos implícitas

que los creadores de noticias quieren plasmar en el texto, este es el núcleo principal de las noticias falsas, ya que contiene todas las ideas, sentimientos y puntos de vista importantes que los autores quieren transmitir a los lectores.

Contexto social

El contexto social se refiere a toda la actividad que tiene el usuario dentro del sistema y el entorno social en el que opera la difusión de la noticia, incluye cómo se distribuyen los datos sociales y cómo los usuarios interactúan entre sí. Actualmente, la forma de compartir y difundir la información está cada vez más dominada por la interactividad de la tecnología en redes sociales.

Figura 2.2: Ejemplo de una noticia falsa de estilo sátira haciendo referencia a sucesos de la política ecuatoriana

2.1.2. Detección basada en el conocimiento

Acorde con Zhou, Zafarani (2020) y Zhang, Ghorbani (2019) en la detección de noticias falsas desde un enfoque del conocimiento, usualmente se utiliza un proceso conocido como *fact-checking*, este fue desarrollado inicialmente por periodistas, y actualmente es usado por gran parte de los medios de comunicación. El fact-ckecking tiene como objetivo verificar la autenticidad de la información, comparando el conocimiento extraído del contenido de las noticias por verificar (por ejemplo, sus

afirmaciones o declaraciones) con hechos conocidos. Tanto criterios de evaluación y métricas visuales, se utilizan para determinar el nivel de veracidad de las noticias en el proceso de fact-checking

Fact-checking manual

El proceso de fact checking usualmente es realizado por un selecto grupo de profesionales denominados *fact checkers*, de gran credibilidad ya que esto conduce a resultados altamente precisos, sin embargo este proceso puede requerir bastante tiempo de ejecución con un costo alto de manutención, además presenta dificultades cuando el contenido de información a verificar es masivo. Algunos de los servicios más populares en inglés de fact checking basado en expertos tenemos: *PolitiFact*[1], *FactCheck*[2], *Snopes*[3], entre otros. En el caso ecuatoriano hay una organización que es el primer medio dedicado a la verificación del discurso público y los contenidos engañosos que circulan en internet establecido en el año de 2016 por iniciativa de la organización Fundamedios: *Ecuador Chequea*[4].

Otra forma de realizar fact checking es la verificación de datos a través de fuentes colectivas, la cual se basa en una gran población de individuos regulares actuando como verificadores de hechos. Comparado con el fact checking basado en expertos, esta es relativamente difícil de administrar, menos creíble y precisa debido al sesgo político de sus verificadores, pero teniendo una mejor escalabilidad, que en ocasiones también es insuficiente para cubrir el inmenso volumen de noticias falsas que circulan a diario.

Fact-checking automático

La verificación manual de hechos no se adapta al volumen de información que se crea cada día en linea, especialmente en redes sociales. Para abordar la escalabilidad, se han desarrollado técnicas de verificación automática de hechos, que se basan en gran medida en la extracción de información mediante el procesamiento del lenguaje natural (NLP) y técnicas de aprendizaje automático o machine learning. Los trabajos desarrollados respecto al fact-checking automático se analizará posteriormente en la **Sección 2.2**.

[1]http://www.politifact.com/
[2]https://www.factcheck.org/
[3]https://www.snopes.com/
[4]https://www.ecuadorchequea.com/

2.1.3. Extracción de información

Siguiendo con lo analizado por Zhang y Ghorbani (2019), se presentan las características utilizadas para la detección automática de noticias falsas en la práctica. En base a lo mencionado en la **Sección 2.1.1** existen tres tipos principales de conjuntos de características, y en cada conjunto hay diferentes categorías.

Características basadas en el usuario

Este tipo de características son ampliamente usadas para la detección de cuentas de usuario sospechosas ya que capturan las características únicas de cuentas sospechosas o cuentas no humanas, se clasifican de la siguiente manera:

- *Perfil de usuario:* Aquí se incluye la información básica del usuario, como el nombre de la cuenta, la información de geolocalización, los datos de registro del usuario, verificados o no, tienen descripción o no, datos demográficos, etc.
- *Credibilidad del usuario:* Este tipo de características registran el impacto y la credibilidad de la cuenta, por lo que incluye el número de contactos del usuario, el radio entre a los contactos que sigue y los contactos que siguen a esa cuenta, puntuación de credibilidad del usuario, tipo de contenido que comparte, entre otros.
- *Comportamiento del usuario:* El comportamiento es parte de las características basadas en el contexto social, y tienen como objetivo obtener un patrón de comportamiento del usuario tanto para usuarios engañosos como para usuarios legítimos. Por ejemplo, la cantidad de publicaciones que realiza una cuenta en una ventana de tiempo.

Características basadas en el contenido de noticias

Las características basadas en el contenido de las noticias son atributos clave para la identificación de noticias falsas, ya que son las principales fuentes de extracción de información y son ampliamente usados en la práctica para el análisis de representación y detección desde un enfoque de minería de datos.

- *Lingüísticas y sintácticas:* Se refieren al componente fundamental, estructura

gramatical y sintáctica del *lenguaje natural* [5]. Técnicas como *bag-of-words*, *n-gram*, *Frecuencia de término (TF)*, *Frecuencia de término inversa (TF-IDF)* son las características mayormente empleadas para el procesamiento del lenguaje a un nivel de análisis palabra por palabra. También se puede usar la presencia de caracteres especiales, palabras estilísticas, palabras de sentimiento y legibilidad de las palabras.

Para un nivel de análisis por oraciones se incluye el etiquetado de partes del discurso (POS), el promedio de de la longitud de las publicaciones, la frecuencia de las puntuaciones y frases que se repiten en una oración, la polaridad promedio de la oración (positiva, negativa o neutra) y complejidad de la oración.

Ahora más general, las características a un nivel de contenido se refieren a los atributos de la información sin procesar de las noticias, en estas se incluyen: la puntuación de sentimiento de una noticia en general (la identificación de la intensidad positiva o negativa en un texto), la postura de una publicación (acuerdo o desacuerdo), temática de la noticia, numero de símbolos y caracteres especiales y links externos dentro de la noticia.

- *Basadas en el estilo*

  Como el título lo sugiere, estas características tienen como objetivo representar el estilo de escritura de los autores de noticias falsas, debido a que los creadores de estas noticias tratan en general replicar el estilo de escritura de una nota real para que la tarea de distinguir entre noticias falsas y noticias reales resulte más difícil para el lector. Por ejemplo, los patrones de edición del creador de la noticia (número de correcciones, pulsaciones de las teclas, clicks hechos con el mouse), tiempo que tarda en escribir el texto, el tiempo medio de escritura de cada palabra y el tiempo medio de escritura entre palabras.

- *Visuales*

  Las imágenes o videos que se incluyen en el contenido de las noticias contienen información de vital importancia acerca de artículos sospechosos de ser noticias falsas, se considera no solo aspectos como el número de imágenes y videos dentro de la noticia, también se considera la claridad, coherencia, proporción de las imágenes, histogramas de similitud, relación entre imágenes, y más.

[5] Lenguaje que se ha desarrollado y ha evolucionado de forma natural como medio de comunicación entre las personas (Collins English Dictionary, s.f., definición 1)

Características basadas en el contexto social

Las características basadas en el contexto social están diseñadas para representar los patrones en como se esparcen las noticias en internet y la interacción que existe entre usuarios. Se puede ver a estas características desde un enfoque de red, distribución y temporal.

- *Basadas en la red*

  Se enfoca en el análisis dentro de una red, intenta centrarse en grupos de usuario similares, desde diferentes perspectivas como ubicación, antecedentes educativos y hábitos. Estas características se extraen de redes de usuarios en específico, mencionando algunos ejemplos se tiene: redes de postura, concurrencia, amistades y difusión. La información extraída de las redes puede ser utilizada para encontrar características únicas de cierto tipo de red y analizar la similitud o diferencia entre los usuarios.

- *Distribución*

  Las características basadas en la distribución pueden ayudar a la distinción de los patrones en como se difunden las noticias falsas en linea, por lo general, se construyen árboles de propagación para facilitar la caracterización de la naturaleza en como se esparcen, pero también se usan características como el número de réplicas del artículo original, la fracción de reposts que hace una cuenta en especial, el nivel de popularidad y cuan sospechosa es de ser una noticia falsa.

- *Temporales*

  Las características temporales se utilizan para describir el comportamiento de publicación del creador de una noticia falsa en forma de una serie de tiempo, son excelentes atributos para detectar actividades de publicaciones sospechosas en una ventana de tiempo, se pueden utilizar de igual forma para indicar el nivel de falsedad de la información. Las características basadas en el tiempo más comunes se incluye el intervalo entre dos publicaciones, la frecuencia de publicación, respuesta y comentario de una cuenta determinada, la hora del día en que se publica, comparte y comenta la información original, y el día de la semana en que la información es publicada.

2.2. Trabajos relacionados

Como hemos visto en la **Sección 2.1.2**, el proceso de verificación de hechos o fact-checking se lo ha ido realizando de manera manual, pero dado la masiva cantidad de noticias falsas en internet, especialmente en redes sociales, se han desarrollado técnicas capaces de abordar el volumen de información a través de una detección automática utilizando características extraídas de las noticias, como se describió en la **Sección 2.1.3** y algoritmos de machine learning, que serán analizados a detalle en la **Sección 3.2**. Para esta sección se presentan algunos trabajos de interés relacionados con la detección de noticias falsas correspondiente al proceso de fact-checking automático.

3HAN: A Deep Neural Network for Fake News Detection

Singharia, Fernandez y Rao (2017) presentan un detector automatizado basado en deep learning a través de una red neuronal de atención jerárquica de tres niveles (3HAN): palabras, oraciones y encabezados, para la detección precisa de noticias falsas. Su modelo se basa en la representación de un articulo de noticia como un *vector de noticia*, el que es utilizado para la clasificar un artículo asignando una probabilidad de ser falso, 3HAN proporciona una puntuación de importancia para cada palabra y oración de un artículo en función de su relevancia para retornar dicha probabilidad. La arquitectura de 3HAN se muestra en la **Figura 2.3**

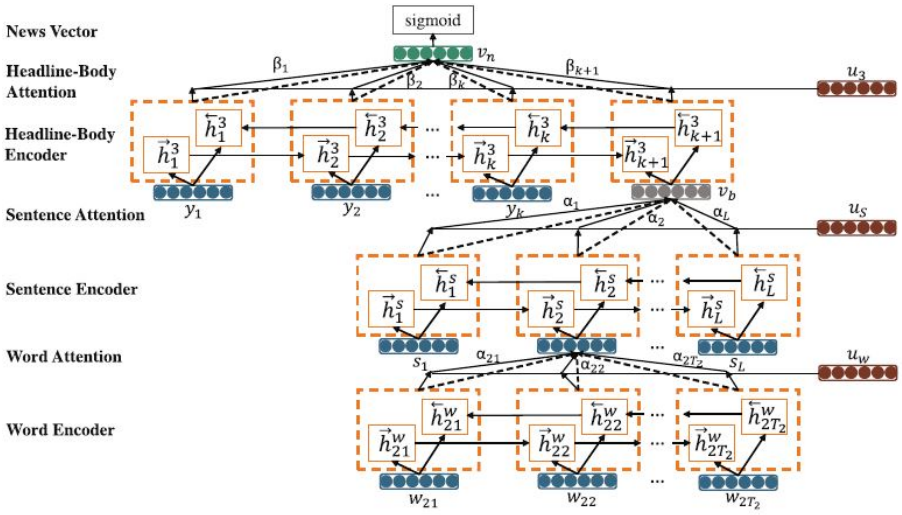

Figura 2.3: Arquitectura de la red neuronal utilizada para el modelo 3HAN (Singharia, Fernandez y Rao, 2017, p. 574)

Los datos utilizado para el entrenamiento de este modelo son de carácter lin-

güístico a un nivel de palabras y oraciones, extraidos de la plataforma *PolitiFact*[6] para el conjunto de noticias falsas y del listado de sitios verificados populares de EE.UU proporcionado por *Forbes* [7] para el conjunto de noticias reales. Los conjuntos de noticias pertenecen al periodo de elecciones presidenciales en EE. UU del año 2016. 3HAN en base a sus experimentos se logra una excelente precisión del 96,77 % superando otros modelos basados en redes neuronales.

Computational Fact Checking from Knowledge Networks

Ciampaglia et al. (2015) presentan un modelo basado en redes o grafos, en el que la verificación de hechos realizadas en el proceso de fact-checking manual puede aproximarse bastante bien al encontrar el camino más corto entre nodos de conceptos bajo métricas de proximidad semántica adecuadamente definidas sobre grafos de conocimiento, este enfoque es viable con técnicas computacionales eficientes.

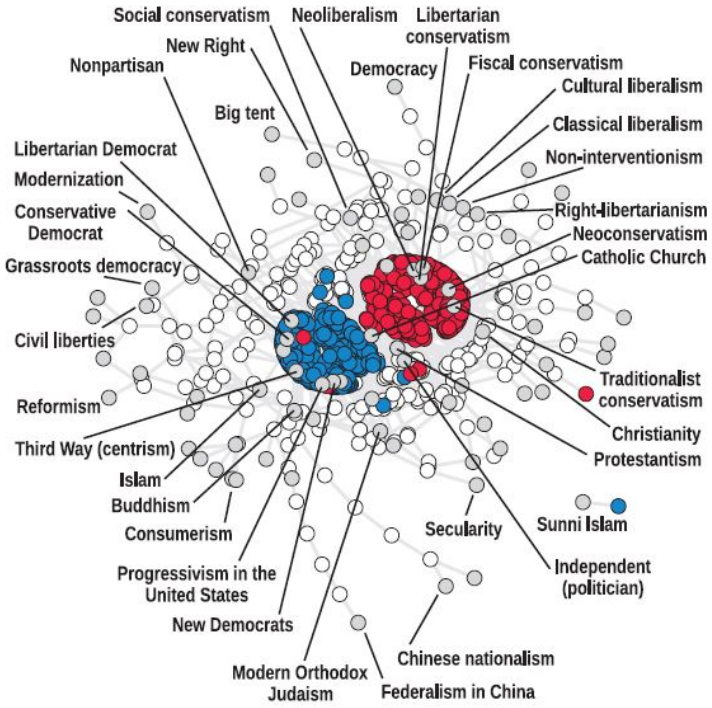

Figura 2.4: Inferencia de la afiliación partidista de los congresistas estadounidenses. Los nodos rojos y azules corresponden a miembros del Congreso, los nodos grises a ideologías y nodos blancos a vértices de cualquier otro tipo (Posadas et al.,2019, p.5)

En este modelo un grafo de conocimiento se produce de la unión de los nodos que denotan afirmaciones (sujetos u objetos de declaraciones) y las aristas que deno-

[6]http://www.politifact.com/

[7]Es una compañía de medios global, que se enfoca en negocios, inversiones, tecnología, emprendimiento, liderazgo y estilo de vida.

tan conectores lingüísticos. Para verificar si una afirmación es cierta, se espera que exista una arista del grafo de conocimiento que conecte con la entidad, o si existe el camino más corto entre entidades del grafo de conocimiento. Por otro lado, si la afirmación es falsa, no debe haber aristas ni caminos cortos que conecten a dicha afirmación.

En la creación del modelo los datos utilizados fueron extraidos de *Wikipedia* [8], en los que constan todas las declaraciones fácticas extraídas de las cajas de información de Wikipedia, creandose así un grafo de conocimiento con 3 millones de entidades unidas por 23 millones de aristas aproximadamente. En base a los experimentos realizados los mejores resultados obtenidos es para una representación con grafos no dirigidos evaluando las clasificaciones mediante dos algoritmos de clasificación: vecinos más cercanos y árboles aleatorios, llegando a niveles de precisión que sobrepasan el 95 % aproximadamente

Detection of fake news in a new corpus for the Spanish language

Posadas-Durán et al. (2019) presentan un modelo para analizar y detectar información engañosa, presentes en gran cantidad de sitios web en idioma español. Para la creación del modelo se usó un conjunto de datos de noticias recopiladas manualmente de distintos sitios web para crear un nuevo corpus de noticias etiquetadas como noticias falsas y reales con el objetivo de su detección automática, también se proporciona relacionado a la temática de la noticia: ciencia, deportes, economía, educación, entretenimiento, política, salud, seguridad y sociedad.

El entrenamiento se lo realizó siguiendo el proceso para un modelo de aprendizaje supervisado, particionando el conjunto de datos en entrenamiento y test, 70 % y 30 % respectivamente, y utilizando los siguientes algoritmos de clasificación: *máquinas de soporte vectorial*, *regresión logística*, *bosques aleatorios* y *Boosting*, evaluando el desempeño para dos escenarios: removiendo palabras repetitivas y considerando palabras repetitivas. La extracción de información y su representación se la realizó a través de características lingüísticas obtenidas de tres técnicas: *bag-of-words*, *n-grams* y POS *tags n-grams*. En base a los experimentos realizados los algoritmos llegan a niveles de precisión cercanos al 75 %.

[8]Enciclopedia en línea gratuita, creada y editada por voluntarios de todo el mundo y alojada por la Fundación Wikimedia

Capítulo 3

Marco Teórico

En la primera sección de este capítulo se presenta las definiciones y conceptos necesarios para entender el proceso de aprendizaje estadístico, desde su definición hasta la explicación del caso particular de los problemas de clasificación. La segunda y tercera sección se basan en la explicación de la estructura y el funcionamiento de cada uno de los modelos de clasificación utilizados durante este trabajo y el método de evaluación que se sigue para elegir el mejor modelo. Finalmente se hace una revisión de las métricas usuales que miden la calidad de las predicciones para modelos de clasificación.

3.1. Aprendizaje estadístico

El aprendizaje estadístico remonta sus origenes al trabajo desarrollado por Legendre con influencias teóricas de Gauss cerca del año de 1800, en el se habla sobre el método de *mínimos cuadrados* que es la base de lo que conocemos hoy en día como *análisis de regresión*. Para el año de 1936 Fisher propone un método para seperar un conjunto de datos con dos clases diferentes mediante un hiperplano, este método es denominado como *análisis de discriminante*.

Perceptron (Rosenblatt, 1962) es el primer modelo que se conoce en la historia que hable de *aprendizaje de máquina*. Rosenblatt lo describió como un programa para computadoras el cual pudo demostrar mediante experimentos simples que su modelo podía ser generalizado.

Dado un vector de entrada $x \in \mathbb{R}^n$ y una variable respuesta $y \in \{-1, 1\}$, la relación de dependencia que siguen está dada por la siguiente expresión:

$$y = sign\{w \cdot x - b\}$$

Este modelo está basado en el *análisis de discriminante* de Fisher, ya que busca mediante un hiperplano determinado por el vector de pesos $w \in \mathbb{R}^n$ y el valor $b \in \mathbb{R}$ separar datos de dos distintas categorías. El objetivo es encontrar estimaciones para w y b, tales que resuelvan el problema.

El modelo perceptron es considerado como el punto de partida del desarrollo de los procesos de aprendizaje estadístico dentro de la inteligencia artificial que junto al poder computacional proporcionado por los lenguajes de programación, se tiene el despunte tecnológico de los últimos años respecto a la analítica de datos. Entender el concepto de aprendizaje estadístico es la pieza fundamental para el desarrollo de este trabajo, motivo por el cual las siguientes secciones se dedican a su descripción y análisis en detalle.

3.1.1. Definición

Acorde con James, Witten, Hastie y Tibshirani (2013), a partir de un conjunto de predictores X y una variable respuesta Y, se asume que existe una relación entre el conjunto de predictores y esta variable respuesta. En general, esta relación es desconocida y busca ser estimada a través de los datos observados, por tanto, el aprendizaje estadístico se refiere a la colección de métodos que se usan para estimar dicha relación.

La aplicación del aprendizaje estadístico es clave para resolver situaciones que se presentan en distintas áreas de la salud, comunicación, economía, etc. Casos de estudio que se pueden ver problemas de aprendizaje estadístico:

- Predicción del nivel de ventas de una empresa usando la información que se tiene sobre el gasto en publicidad de sus productos o servicios.
- Identificación de los factores de riesgo a padecer una enfermedad basados en la información demográfica de las personas
- Clasificación de archivos de texto basándose en la información que estos proveen.

Gran parte de los problemas dentro del aprendizaje estadístico son los de aprendizaje supervisado, en estos, el conjunto de predictores y la variable respuesta están

involucrados en el proceso de entrenamiento, es decir, para cada vector observado x está asociada una observación de la variable y, y así, ajustar un modelo que represente esta relación para realizar predicciones sobre y o conocer mejor la forma en que afectan los cambios de los predictores en la variable respuesta.

3.1.2. Etapas del aprendizaje

Basándose en lo descrito por Mohri, Rostamizadeh y Talwalkar (2018), a continuación se describe las diferentes etapas del aprendizaje estadístico que se siguen para resolver un problema en la práctica.

Se inicia por particionar de manera aleatoria los datos en entrenamiento, validación y test. El conjunto de entrenamiento está destinado a que el modelo realice el proceso de aprendizaje por medio de los ejemplos que se le proporciona. Posteriormente para elegir los mejores hiperparámetros [1] del modelo entrenado, se evalúa el desempeño que este tiene al resolver el problema en el conjunto de validación variando los hiperparámetros, y así, se elige los que ofrezcan los mejores resultados. Una vez que el modelo es entrenado y se escogieron los hiperparámetros óptimos, se usa el conjunto de test con el fin de evaluar el desempeño del modelo al resolver el problema desde un punto de vista general con datos que no intervienen en el proceso de aprendizaje. El tamaño de cada uno de estos conjuntos de datos depende del modelo, pero por regla general el conjunto de entrenamiento debe ser más grande que los de validación y test. Seleccionar los predictores que serán utilizados para entrenar el modelo es crítico ya que una buena elección de los predictores que se usarán para el aprendizaje potenciarán los resultados que se obtengan, por otra parte, una mala elección de los predictores puede guiarnos a resultados engañosos, este proceso depende de la experiencia del investigador sobre el problema.

Una vez descrito las etapas del aprendizaje, hace falta describir el tipo de problemas que es interés de este trabajo, ya que el objetivo de este trabajo es identificar noticias falsas, y esto se lo realizara clasificando noticias como verdaderas o falsas .

[1] Parámetros que sirven para controlar la implementación del modelo y el proceso de aprendizaje, mas no son los parámetros que definen algoritmo de aprendizaje utilizado para la creación del modelo

3.2. Algoritmos de clasificación

Dentro del aprendizaje supervisado cuando la variable de respuesta y es *cuantitativa* se cataloga como un problema de *regresión* mientras que, en el caso en que y sea cualitativa se denomina problema de *clasificación*. Varias situaciones se pueden ver como problemas de clasificación.

- Clasificar imágenes de mascotas (perro, gato, loro) utilizando la información que proveen los píxeles.
- Predecir la condición médica (sobredosis de narcóticos, accidente cerebro vascular, ataque epiléptico) de un paciente de emergencias por sus síntomas
- Predecir el comportamiento de pago de un cliente (buen pagador, mal pagador) analizando sus información demográfica y actividad bancaria

Identificar noticias falsas encaja dentro del grupo de problemas de clasificación, ya que se va a predecir el tipo de noticia (real o falsa) en base a la estructura del texto desde el enfoque del procesamiento del lenguaje natural. A continuación se describe los modelos de clasificación supervisada usados para la creación de los modelos de este trabajo, las nociones y conceptos presentados fueron tomados de James, Witten, Hastie y Tibshirani (2013) y Hastie, Tibshirani y Friedman (2009)

Modelos aditivos generalizados

Los *modelos aditivos generalizados* proveen una extensión del modelo lineal estándar permitiendo funciones no lineales para cada una de los predictores pero manteniendo la idea de aditividad ya que varias situaciones en la vida real no se las puede modelizar linealmente. Los modelos aditivos generalizados pueden ser empleados igualmente para resolver tanto problemas de regresión como de clasificación. El modelo aditivo generalizado tiene la siguiente forma

$$g(\mu(X)) = \alpha + f_1(X_1) + ... + f_p(X_p) \tag{3.1}$$

donde $\mu(X) = E(Y|X_1, ..., X_p)$ es la media de la variable de respuesta Y condicionada a X que está relacionada con una función aditiva de los predictores a través de un una función de enlace g y cada f_j son funciones suaves no paramétricas.

- $g(\mu) = \mu$ utilizado para modelos lineales y aditivos para con datos de respuesta gaussiana.
- $g(\mu) = logit(\mu)$ utilizado para modelizar probabilidades de una distribución Binomial
- $g(\mu) = log(\mu)$ utilizado para modelos de reucneto de datos usando la distribución de Poisson

3.2.1. Máquinas de soporte vectorial

Se presenta la idea básica de un clasificador usando un hiperplano basado en el modelo *perceptron* de Rosenblatt, esto nos ayudará a tener una mejor comprensión acerca de que son las máquinas de soporte vectorial, que tipo de problema resuelve y las limitaciones que se presentan.

Hiperplano

En un espacio S de dimensión $n+1$ un hiperplano H es un subespacio afin de dimensión n, que puede ser expresado por la siguiente ecuación

$$\beta_0 + \beta_1 X_1 + ... + \beta_n X_n = 0 \tag{3.2}$$

para $\beta_0, ..., \beta_n \in \mathbb{R}$. Se dice que (3,2) define al hiperplano ya que cualquier $(X_1, ..., X_n)$ que cumpla la expresión estará dentro del mismo y los que no la cumplan estarán en una de las dos regiones (3,3) y (3,3) de S divididas por el hiperplano H.

$$S_1 = \{(X_1, ..., X_n) \in \mathbb{R}^n : \beta_0 + \beta_1 X_1 + ... + \beta_n X_n < 0\} \tag{3.3}$$

$$S_2 = \{(X_1, ..., X_n) \in \mathbb{R}^n : \beta_0 + \beta_1 X_1 + ... + \beta_n X_n > 0\} \tag{3.4}$$

Clasificador de margen máximo

A partir de un conjunto de n vectores de entrenamiento $x_1, ..., x_n \in \mathbb{R}^p$ clasificados en dos clases, es decir, $y_1, ..., y_n \in \{-1, 1\}$ de modo que forman los pares $(x_1, y_1), ..., (x_n, y_n)$, el objetivo es ajustar un hiperplano que separe perfectamente

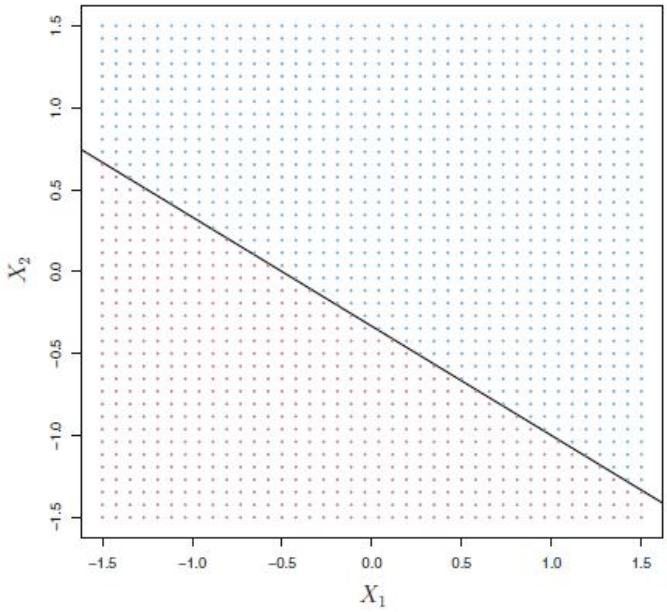

Figura 3.1: Hiperplano 1 + $2X_1$ + $3X_2$ = 0. La región azul es el conjunto de puntos para los cuales 1 + $2X_1$ + $3X_2$>0, y la región lila es el conjunto de puntos para los cuales 1 + $2X_1$ + $3X_2$ <0 (James, Witten, Hastie y Tibshirani, 2013, p. 339)

el espacio de estos vectores de entrenamiento acorde a sus clases. Suponiendo que existe un hiperplano que separe los vectores de entrenamiento, este tendrá las siguiente propiedad para $i = 1, ..., n$

$$y_i(\beta_0 + \beta \cdot x_i) > 0 \tag{3.5}$$

con $\beta = (\beta_1, ..., \beta_p) \in \mathbb{R}^p$. La dificultad radica en que no solo existe un hiperplano que separe los vectores de entrenamiento, existen infinitos que los separan, por tanto, la estrategia es elegir el que ofrezca el *margen* máximo.

La mínima distancia perpendicular que existe entre los vectores de entrenamiento y el hiperplano la denotaremos M y es por esto que el vector o los vectores que estén a una distancia M se los denomina *vectores de soporte*, el espacio a una distancia M del hiperplano es el *margen*. El problema de encontrar el hiperplano de margen máximo puede ser visto como un problema de optimización.

$$\begin{aligned} &\max_{\beta_0, \beta} M \\ &\text{s.a} \quad \|\beta\| = 1 \\ &y_i(\beta_0 + \beta \cdot x_i) \geq M, \quad \forall i \in \{1, ..., n\} \end{aligned} \tag{3.6}$$

Para el caso en que el espacio de vectores de entrenamiento no pueda ser sepa-

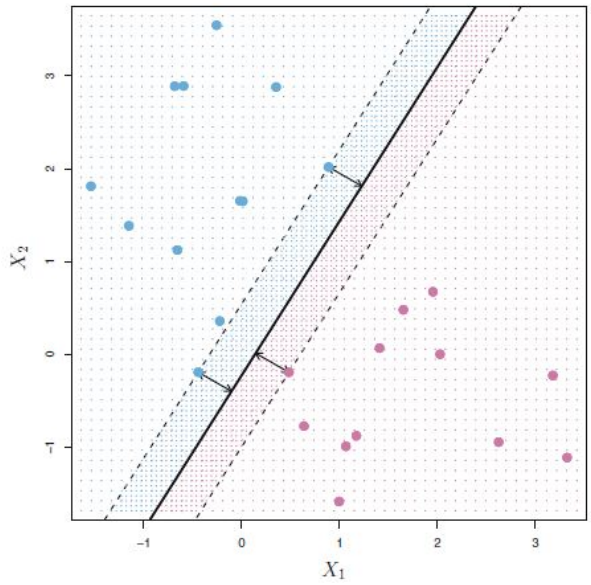

Figura 3.2: El hiperplano de margen máximo se muestra como una línea continua. Los dos puntos azules y el punto lila que se encuentra en las líneas entrecortadas son los vectores de soporte (James, Witten, Hastie y Tibshirani, 2013, p. 342)

rado perfectamente el problema de optimización (3.6) no tiene solución, por lo que se plantea ajustar un hiperplano que no separe perfectamente el espacio, es decir, el hiperplano separará la mayoría de las observaciones correctamente, esto se logra permitiendo que cierto número de observaciones que están dentro de los límites de los márgenes o incluso fuera, estén en lados erróneos del hiperplano.

$$\begin{aligned} &\max_{\beta_0,\beta,\xi_1,\ldots,\xi_n} M \\ &\text{s.a} \quad \|\beta\| = 1 \\ &y_i(\beta_0 + \beta \cdot x_i) \geq M(1-\xi_i), \quad \forall i \in \{1,\ldots,n\} \\ &\sum_{i=1}^{n} \xi_i \leq C \\ &\xi_i \geq 0 \quad \forall i \in \{1,\ldots,n\}, \quad C > 0 \end{aligned} \tag{3.7}$$

$\xi_1, \ldots, \xi_n$ son variables de holgura que permiten que el problema sea más flexible y permita que un número determinado de observaciones estén en lados erróneos del hiperplano, estos indica donde se encuentra la observación en relación al margen, si $\xi_i = 0$ la observación i está dentro de la región correcta del del margen, si $0 < \xi_i \leq 1$ la observación está en la región errónea del *margen* y si $\xi_i > 1$ la observación está en la región errónea del hiperplano. El parámetro de ajuste C juega un papel importante, ya que controla cuantas observaciones estarán mal clasificadas.

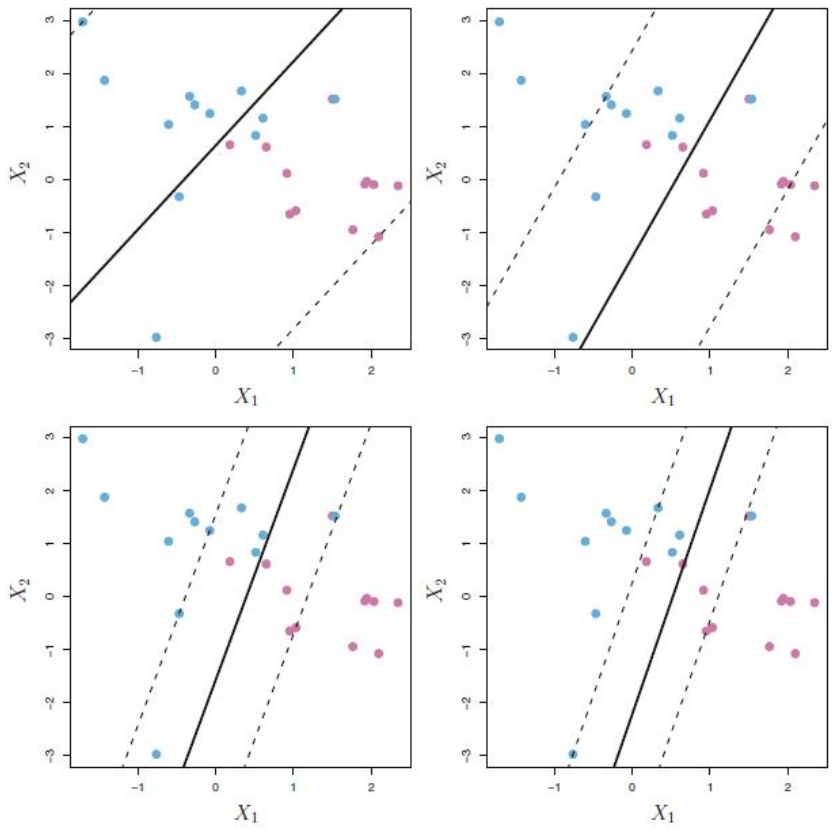

Figura 3.3: Clasificador de vectores de soporte utilizando cuatro valores diferentes de C. El mayor valor de C se utilizó en la parte superior del panel izquierdo, se usaron valores más pequeños en la parte superior derecha, inferior izquierda e inferior derecha. Cuando C es grande, hay más alta tolerancia para que las observaciones sean en el lado equivocado del margen, por lo que este será grande. A medida que C disminuye, la tolerancia para que las observaciones estén en el lado incorrecto del margen disminuye, y el margen se estrecha (James, Witten, Hastie y Tibshirani, 2013, p. 348)

Kernels

Hasta el momento se ha analizado el caso en que el problema puede ser resuelto de forma lineal mediante la separación del espacio de vectores de entrenamiento por un hiperplano. Las máquinas de soporte vectorial son una generalización del problema (3.7) en el que se busca una transformación ϕ que extiende la dimensión del espacio de observaciones de tal manera que este nuevo espacio puede ser separado

por un hiperplano.

$$\begin{aligned} &\max_{\beta_0,\beta,\xi_1,...,\xi_n} M \\ &\text{s.a} \quad \|\beta\| = 1 \\ &y_i(\beta_0 + \beta \cdot \phi(x_i)) \geq M(1-\xi_i), \quad \forall i \in \{1,...,n\} \\ &\sum_{i=1}^{n} \xi_i \leq C \\ &\xi_i \geq 0 \quad \forall i \in \{1,...,n\}, \quad C > 0 \end{aligned} \tag{3.8}$$

con $\phi(x) : \mathbb{R}^p \rightarrow \mathbb{R}^q \quad q \geq p$. Reescribiendo (3.8) equivalentemente como un problema de optimización convexa

$$\begin{aligned} &\max_{\beta_0,\beta} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{s.a} \quad \xi_i \geq 0, \quad C > 0, \quad y_i(\beta_0 + \beta \cdot \phi(x_i)) \geq 1 - \xi_i, \forall i \in \{1,...,n\} \end{aligned} \tag{3.9}$$

Resolviendo para β por multiplicadores de Lagrange se tiene que $\beta = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$ para $0 < \alpha_i < C, \quad i = 1,...,n$. Utilizando esta expresión se puede escribir la regla de clasificación como

$$G(x) = sign\{f(x)\} \tag{3.10}$$

donde,

$$\begin{aligned} f(x) &= \beta_0 + \beta \cdot \phi(x) \\ &= \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \phi(x_i) \cdot \phi(x) \\ &= \beta_0 + \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) \end{aligned} \tag{3.11}$$

No es necesario especificar la transformación ϕ, ya que basta en especificar la forma del producto interno de los vectores transformados en los nuevos espacios, a este producto interno se lo denomina *kernel*.

$$K(x, x') = \phi(x) \cdot \phi(x') \tag{3.12}$$

Puede verse a los kernels como funciones que cuantifican la similitud entre dos observaciones. En la practica los *kernels* mayormente usados son los siguientes

- Kernel lineal: $K(x', x) = x' \cdot x$
- Kernel Polinómico: $K(x', x) = (1 + x' \cdot x)^d$
- Kernel Gaussiano: $K(x', x) = exp(-\gamma \|x' - x\|^2)$
- Kernel Sigmoide: $K(x', x) = tanh(\alpha(x' \cdot x) + \beta)$

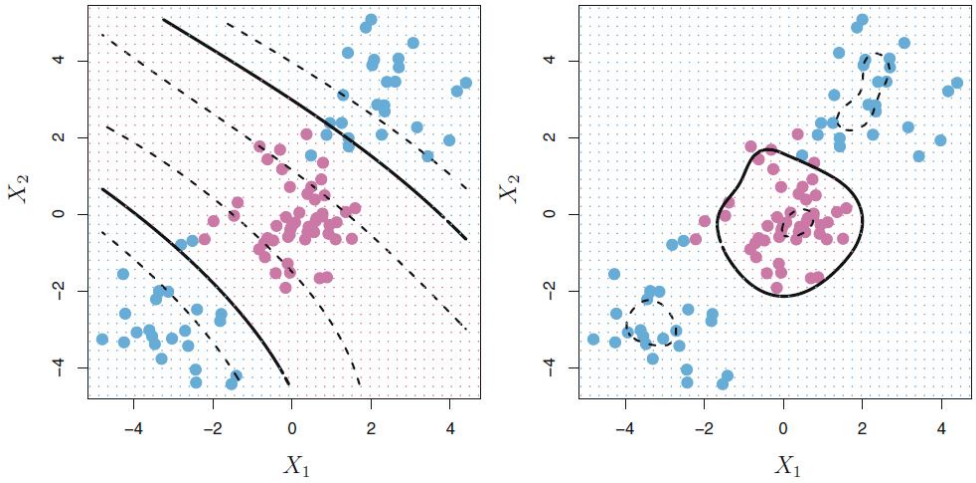

Figura 3.4: Izquierda: Clasificador con kernel polinomial de grado 3. Derecha: Clasificador con kernel gaussiano (James, Witten, Hastie y Tibshirani, 2013, p. 353)

Una de las ventajas de la utilización de kernels es que son mucho menos costosos computacionalmente, ya que para el cálculo del clasificador solo se requiere el calculo de los productos internos de las observaciones del espacio de entrenamiento y no transformar las observaciones al nuevo espacio. En las máquinas de soporte vectorial, la extensión del espacio de entrenamiento puede llegar a ser tan grande que los cálculos se convierten intratables, por lo que la implementación de funciones kernels para representar estas transformaciones es muy importante en la practica.

3.2.2. Bosques aleatorios

Los modelos que involucran la segmentación en regiones simples del espacio de predictores y sus reglas de clasificación se las puede modelizar mediante una serie de decisiones relacionadas que se conocen como modelos basados en *árboles de decisión*.

Arboles de clasificación

Los arboles de decisión son modelos simples que ofrecen resultados potentes, por lo que suelen ser usados para resolver problemas de regresión y clasificación. Estos modelos se basan en crear una partición [2] del espacio de entrenamiento, la clasificación de una observación será la clase más común presente en las observaciones correspondientes a la región al que esta pertenece.

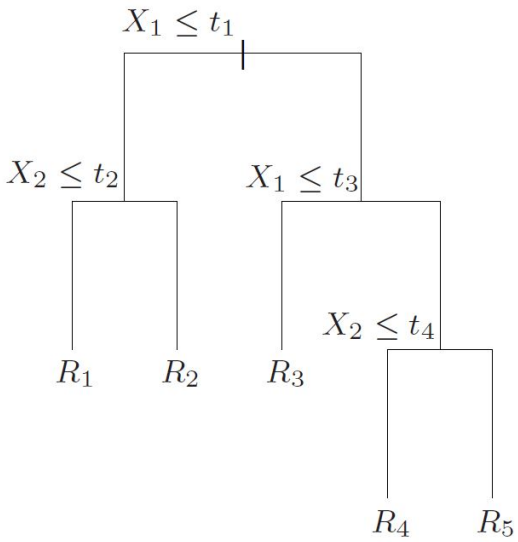

Figura 3.5: Ejemplo de árbol de decisión para dos variables X_1 y X_2 y con cuatro nodos terminales R_1, R_2, R_3, R_4 y R_5. (James, Witten, Hastie y Tibshirani, 2013, p. 308)

Los puntos a lo largo del árbol donde el espacio de predictores está dividido se conocen como *nodos internos* y las lineas que conectan cada uno de los nodos se las denomina *ramas*. Las n regiones $R_1, ..., R_n$ de la partición comúnmente llamados *nodos terminales* u *hojas* son rectángulos n-dimensionales creados por *división binaria recursiva* con el fin de obtener la mejor división del espacio de predictores en cada nodo.

La forma más común de medir la potencia de la división es la tasa de error de clasificación, esta mide la proporción de observaciones que no pertenecen a las clase más común, en la practica se usan medidas más sensitivas como el coeficiente el *ínidce de Gini* o *cross-entropy* ya que estas son una forma de medir la impureza de los

[2] Una partición de un conjunto es una familia de subconjuntos no vacíos, disjuntos dos a dos, cuya unión es todo el espacio.

nodos (3.13) denotados por Q_m

$$\begin{aligned} \text{Tasa de error de clasificación:} \quad & 1 - \max_k\{\hat{p}_{mk}\} \\ \text{Índice de Gini:} \quad & \sum_{i=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}) \\ \text{Cross-entropy:} \quad & -\sum_{i=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk}) \end{aligned} \tag{3.13}$$

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{3.14}$$

donde $\hat{p}_{mk}$ es la proporción de las observaciones de entrenamiento que se encuentran en la región m que pertenecen a la clase k.

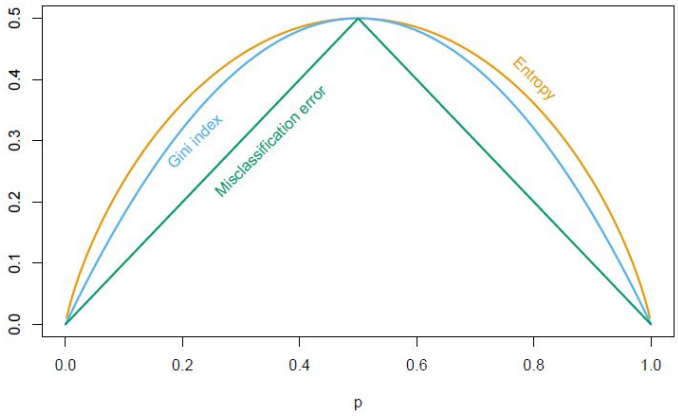

Figura 3.6: Medidas de impurezas de los nodos para la clasificación de dos clases en función de la proporción p. Cross-entropy ha sido escalada para que pase por (0.5,0.5) (Hastie, Tibshirani y Friedman, 2009, p.309)

División binaria recursiva

La división binaria recursiva se empieza tomando un predictor x_i y un punto de corte c_i de tal manera que el espacio de predictores garantice la mejor división del espacio de entrenamiento en dos regiones respecto a las métricas de (3.13), este primer árbol constará solo de dos terminales R_1 y R_2

$$R_1 = \{(x_1, ..., x_n) \in \mathbb{R}^n : x_i \leq c_i\} \quad R_2 = \{(x_1, ..., x_n) \in \mathbb{R}^n : x_i > c_i\} \tag{3.15}$$

Posteriormente se divide una de las regiones resultantes en vez de particionar

nuevamente todo el espacio de predictores, asegurándose de que sea la mejor división obteniendo así tres nodos terminales. Recursivamente dividiendo cada una de las regiones resultantes, se puede seguir haciendo crecer a nuestro árbol de decisión conforme a nuestros requerimientos hasta alcanzar un criterio de paro. Como se mencionó anteriormente, una vez creadas todas las regiones $R_1, ..., R_n$ la clasificación de una observación será la clase más común a la que esta pertenezca.

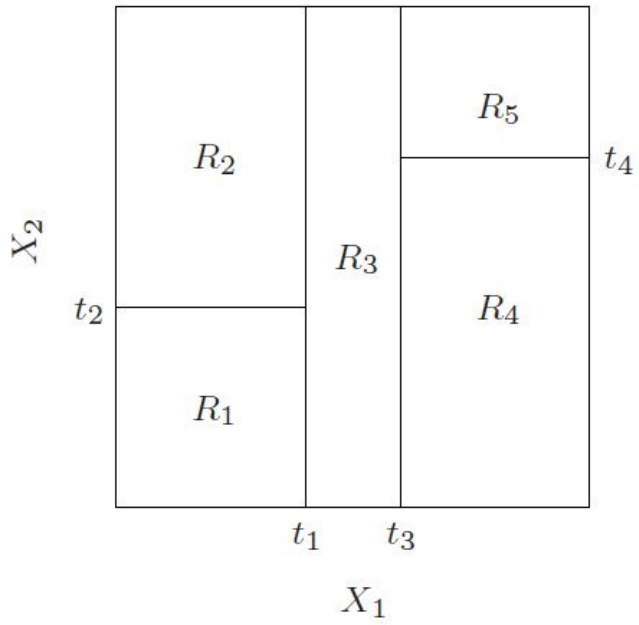

Figura 3.7: Ejemplo de partición del espacio de entrenamiento de dos variables X_1 y X_2 en R_1, R_2, R_3, R_4 y R_5 nodos terminales (James, Witten, Hastie y Tibshirani, 2013, p. 308)

Tree Pruning o poda de árboles

Un árbol con una profundidad muy extensa nos puede llevar a problemas de sobre ajuste del modelo, mientras que un árbol poco extenso no podría capturar la estructura y los patrones dentro de los datos, por lo que la estrategia que se se sigue es crear un árbol T_0 lo más grande posible para luego *podarlo*, con podar se refiere a contraer uno o más de sus nodos internos y obtener un subárbol T. La mejor forma de podar un árbol es utilizar el *criterio costo-complejidad*, el cual se basa en considerar una familia de subárboles indexados por un parámetro de ajuste $\alpha \geq 0$, de modo que para cada valor de α se tiene un subárbol $T \subset T_0$ que minimiza

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m + \alpha|T| \tag{3.16}$$

donde $|T|$ es el número de nodos terminales en T. El parámetro α controla la compensación que hay entre la complejidad del árbol de decisión y como este se ajusta a los datos, es claro que si $\alpha = 0$ el subárbol T será T_0 y a medida que α se incrementa el costo por tener un subárbol con bastantes nodos terminales también lo hace, por esto resulta en un subárbol menos profundo. La estimación del mejor valor para α usualmente se lo realiza mediante validación cruzada.

Bosques aleatorios

La idea básica de los bosques aleatorios está en el proceso de *bagging*, este proceso consiste en ajustar árboles de decisión a partir de B muestras *bootstrap*[3], la diferencia radica en que cada vez que un árbol realiza una división en vez de considerar todo el conjunto de predictores, solo se considera un subconjunto aleatorio de este con el objetivo de reducir la correlación existente entre los árboles.

La cantidad de predictores considerados para la división de árboles aleatorios difiere si se trata de un problema de regresión o clasificación, los autores recomiendan de un conjunto de p predictores

- Para clasificación, el valor de m es $\lfloor \sqrt{p} \rfloor$ y como criterio de parada el tamaño mínimo de un nodo es 1
- Para regresión, el valor de m es $\lfloor \frac{p}{3} \rfloor$ y como criterio de parada el tamaño mínimo de un nodo es 5

Suponiendo que se tiene K clases en el conjunto de entrenamiento, el clasificador para una observación se define como

$$G_B(x) = \underset{k}{\text{argmax}}\, f_B(x) \tag{3.17}$$

donde $f_B(x)$ es un K-vector $(p_1, ..., p_k)$ en que p_k es la proporción de los B árboles que clasifican a la observación x en la clase K, en pocas palabras el G elige la clase más recurrente de los B árboles entrenados.

En promedio para el entrenamiento de cada árbol del bosque aleatorio se usan unas dos terceras partes de los datos de entrenamiento, las observaciones restantes que no participan en este proceso se las denomina observaciones *out-of-bag* (OOB).

[3]Muestras extraídas aleatoriamente con reemplazo de los datos de entrenamiento, cada muestra del mismo tamaño que el conjunto de entrenamiento original

Para predecir la variable respuesta de estas muestras, simplemente se promedian las predicciones (en caso de problemas de regresión) o se elige la clase más recurrente (en caso de clasificación) de los árboles de decisión ajustados de la muestra bootstrap a la que estas observaciones no pertencen, de esta forma se obtienen predicciones para cada una de estas observaciones OOB para las cuales se puede calcular el error cuadrático medio en caso de regresión o el error de clasificación para problemas de clasificación, y una vez que se estabilice el entrenamiento puede concluirse.

Algoritmo 1: Random Forest para clasificación

Para $b = 1$ **hasta** B **:**

(a) Tomar una muestra bootstrap Z de tamaño N del espacio de entrenamiento

(b) Ajustar un arbol T_b de la muestra Z en el que cada división del árbol se lleva a cabo con lo siguientes pasos

1. Seleccionar m variables de las p variables
2. Tomar la mejor variable de las m seleccionadas para dividir el árbol
3. Dividir el nodo

Salida: Colección de árboles $\{T_b\}_1^B$

La clasificación para una observación x será la clase que mas se repite en las predicciones de los árobles $\{T_b\}_1^B$

3.2.3. Boosting

Similar a los bosques aleatorios, los modelos boosting se basan en la utilización de una serie de modelos *débiles*[4] que en conjunto proveen un gran poder predictivo y son ampliamente usados para problemas de regresión y clasificación. A diferencia de los bosques aleatorios, los árboles de decisión entrenados no son sobre muestras boostrap, en su lugar, se los entrena sobre modificaciones secuenciales del espacio de entrenamiento, cada árbol que se ajusta usa la información de sus predecesores.

AdaBoost

Dentro de los modelos boosting el modelo propuesto por Freund and Schapire en 1997 denominado *AdaBoost.M1* es uno de los más populares. Para este modelo se tiene un problema de clasificación en que la variable respuesta $Y \in \{-1, 1\}$, el espa-

[4]Un clasificador débil es aquel cuya tasa de error es solo ligeramente mejor que adivinar al azar

cio de vectores de entrenamiento X y un clasificador G. El error de entrenamiento esta dado por

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i)) \tag{3.18}$$

El objetivo de los modelos boosting es aplicar repetidamente un clasificador débil a modificaciones del conjunto de entrenamiento con lo que se generarán M clasificadores débiles nuevos G_m y serán estos los que se usen para la predicción final. La clasificación de una observación x será la combinación ponderada de las predicciones de estos clasificadores débiles.

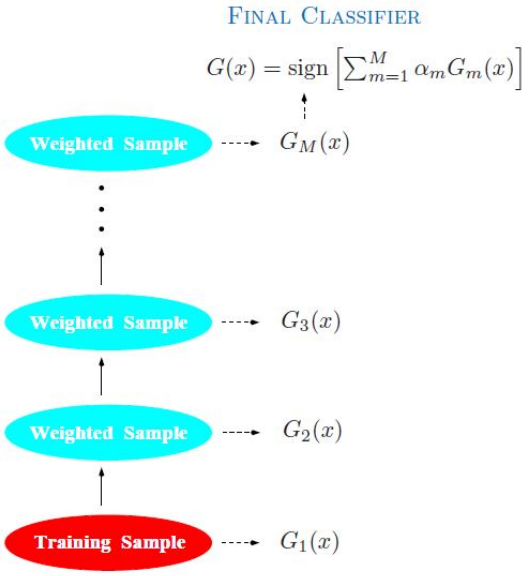

Figura 3.8: Esquema del algoritmo AdaBoost. Los clasificadores se entrenan en modificaciones del conjunto de entrenamiento, y luego se combinan para producir una predicción final (Hastie, Tibshirani y Friedman, 2009, p.338)

$$G(x) = sign\left\{\sum_{m=1}^{M} \alpha_m G_m\right\} \tag{3.19}$$

donde α_m controla la contribución que aporta el predictor G_m al modelo final, dando más relevancia a los que sean más precisos. Las modificaciones que se realizan en el conjunto de entrenamiento consisten en aplicar pesos $w_1, ..., w_N$ a las N observaciones $x_1, ..., x_N$, los pesos se lo inicializa con $w_i = \frac{1}{N} \quad \forall i = 1, .., N$ y conforme pasan las iteraciones, los pesos se van modificando uno por uno haciendo que el algoritmo de clasificación use los nuevos datos ponderados.

En cada iteración las observaciones que fueron erróneamente clasificadas incre-

Algoritmo 2: AdaBoost M1

Inicializar $w_i = \frac{1}{N}, \forall i = 1, .., N$

Para $m = 1$ **hasta** M **:**

(a) Ajustamos el clasificador G_m al conjunto de entrenamiento con los pesos w_i

(b) Calculamos

$$err_m = \frac{w_i \sum_{i=1}^{N} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

(c) Calculamos

$$\alpha_m = log\left(\frac{1 - err_m}{err_m}\right)$$

(d) Hacer $w_i \leftarrow w_i \cdot exp(\alpha_m I(y_i \neq G_m(x_i))), \quad i = 1, ..., N$

Salida: $G(x) = sign\{\sum_{m=1}^{M} \alpha_m G_m(x)\}$

mentan sus pesos, mientras que las que fueron correctamente clasificadas su peso se reduce, así, a medida que el proceso avanza las observaciones que tienen problemas al ser clasificadas adquieren más relevancia para las siguientes iteraciones, obligando a los siguientes clasificadores concentrarse en aquellas que no obtuvieron una predicción precisa por los clasificadores anteriores.

Boosting como modelo aditivo

Los modelos boosting son una forma de ajustar una expansión aditiva de un conjunto de funciones *bases* elementales, el clasificador de AdaBoost.M1 es un caso particular en que las funciones bases son los clasificadores individuales $G_m \in \{-1, 1\}$. La forma general viene dada por la siguiente expresión.

$$f(x) = \sum_{m=1}^{M} \beta_m b(x, \gamma_m) \tag{3.20}$$

con $\beta_m, m = 1, ..., M$ los coeficientes respectivos de expansión de las funciones multivariantes b caracterizadas por el conjunto de parámetros γ_m, la utilidad de la expansión con funciones base se ha utilizado para redes neuronales, procesamiento de señales, regresión con splines y árboles de decisión, para este último caso los γ_m caracterizan las variables de división, los puntos de división en los nodos internos, y las predicciones en los nodos terminales.

Para el **Algoritmo 3** se emplea el uso de las *funciones de pérdida* que serán explicadas con mayor profundidad en la sección 3.3. Podemos expresar el **Algoritmo**

Algoritmo 3: Modelo aditivo progresivo por etapas

Inicializar $f_0 = 0$

Para $m = 1$ **hasta** M **:**

(a) Calcular

$$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i, \gamma)))$$

(b) Hacemmos

$$f_m(x) = f_{m-1}(x) + \beta_m b(x, \gamma_m)$$

Salida: $f(x) = \sum_{m=1}^{M} \beta_m b(x, \gamma_m)$

3 equivalentemente como el algoritmo AdaBoost.M1 mediante la utilización de la función de pérdida exponencial.

$$L(y, f(x)) = exp(-yf(x)) \tag{3.21}$$

En el algoritmo AdaBoost las funciones base b son los clasificadores individuales $G_m \in \{-1, 1\}$ que usando la función de pérdida (3.21) se debe resolver lo siguiente

$$\begin{aligned}(\beta_m, G_m) = \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} exp(-y_i(f_{m-1}(x_i) + \beta G(x_i))) = \\ \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} exp(-\beta y_i G(x_i)))\end{aligned} \tag{3.22}$$

con $w_i^{(m)} = exp(-y_i f_{m-1}(x_i))$, ya que estos pesos no dependen ni de β ni G, precisamente estos son los que se usarán para ponderar las observaciones de entrenamiento, es importante notar para cada iteración del algoritmo los pesos irán cambiado, siendo para la iteración m el valor del peso dependerá de f_{m-1}. La solución para (3.22) se procede por dos pasos. Primero para un valor β dado, la solución para G_m es

$$G_m = \underset{G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) \tag{3.23}$$

esto puede verse mucho más claro si se expresa (3.22) como

$$(e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} \tag{3.24}$$

poniendo el G_m que resuelve (3.22) y resolviendo para β se obtiene.

$$\beta_m = \frac{1}{2} log \left(\frac{1 - err_m}{err_m} \right) \tag{3.25}$$

De modo que $f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$ entonces los pesos se modifican en la siguiente iteración

$$w_i^{(m+1)} = w_i^{(m)} \cdot exp(-\beta_m y_i G_m(x_i)) \tag{3.26}$$

ahora, dado que $-y_i G_m(x_i)$ se puede reescribir como $2I(y_i \neq G_m(x_i)) - 1$, (3.26) se convierte en

$$w_i^{(m+1)} = w_i^{(m)} \cdot exp(2\beta_m I(y_i \neq G_m(x_i))) exp(-\beta_m) \tag{3.27}$$

Haciendo $\alpha_m = 2\beta_m$ y dado que el factor $exp(-\beta_m)$ multiplica a todas las observaciones, este no tiene ningún efecto en los pesos, considerando lo anterior, (3.27) es equivalente a la actualización de los pesos en el **Algoritmo 2**.

Árboles Boosting

Recordando que los árboles de decisión particionan el espacio de entrenamiento en $R_1, ..., R_J$ regiones denominados nodos terminales, ya sea para problemas de regresión o clasificación se asigna una constante $\gamma_j, \quad j = 1, ..., J$ para cada uno de los nodos terminales, por lo que la regla de predicción se la puede formular equivalente como

$$x \in R_j \rightarrow f(x) = \gamma_j \tag{3.28}$$

Por (3.28) un árbol de decisión escrito formalmente sería

$$T(x, \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j) \tag{3.29}$$

donde $\Theta = \{R_j, \gamma_j\}_1^L$. La estimación de estos parámetros se la realiza minimi-

zando

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j) \tag{3.30}$$

para los parámetros R_j y γ_j, $\quad j = 1, ..., J$, con L la función de *costo* respectiva al problema que se esta resolviendo (clasificación o regresión). En la seccion 3.2.2 se habló como encontrar la mejor división para un árbol de decisión e identificar las regiones $R_1, ..., R_j$, el modelo de árboles boosting es la suma de M árboles de decisión ajustados sobre las modificaciones secuenciales del espacio de entrenamiento dados por Θ_m

$$f_M(x) = \sum_{m=1}^{M} T(x, \Theta_m) \tag{3.31}$$

Así, en el **Algoritmo 3** en cada etapa del proceso de entrenamiento se debe encontrar solución a lo siguiente

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m)) \tag{3.32}$$

con $\theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$. Para un problema de clasificación de dos clases y utilizando la función de pérdida exponencial, el enfoque por etapas da lugar al método AdaBoost para impulsar los árboles de clasificación. Como se vio anteriormente la solución es el árbol que minimiza $\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq T(x_i, \Theta_m))$ con pesos $w_i^{(m)} = e^{-y_i} f_{m-1}(x_i)$ y restringido a que $\gamma_{jm} \in \{-1, 1\}$. Sin esta restricción y haciendo uso de la función de pérdida exponencial se tiene

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} exp(-y_i T(x_i, \Theta)) \tag{3.33}$$

Para este caso la división óptima de los árboles de decisión no se realiza usando los criterios de pureza de los nodos vistos en la sección 3.2.2, en su lugar, se dividen los árboles utilizando el criterio de pérdida exponencial ponderada (3.33). Por tanto, dadas las regiones R_{jm} la solución para (3.32) es la siguiente

$$\hat{\gamma}_{jm} = \frac{1}{2} log \left\{ \frac{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = 1)}{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = -1)} \right\} \tag{3.34}$$

3.2.4. Naive Bayes

El modelo presentado pretende calcular la probabilidad a posteriori de las clases de Y dado que existe la información de los predictores, por lo que el uso del teorema de Bayes es fundamental. La utilización de estos modelos son apropiados cuando la dimensión del espacio de entrenamiento es grande ya que facilita el cálculo computacional.

Clasificador

Para un espacio de entrenamiento X representado por p predictores la variable de respuesta Y puede tomar K clases distintas, utilizando el teorema de Bayes para una observación x se tiene lo siguiente

$$p_k(x) = P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{i=1}^{K} \pi_i f_i(x)} \tag{3.35}$$

donde $\pi_k = P(Y = k)$ es la probabilidad a priori de que una observación tomada al azar pertenezca a la clase k, $f_k(X) = P(X|Y = k)$ denota la función de densidad de X para observaciones provenientes de la clases k. Este modelo asume que dada una clase k las distribuciones marginales de los predictores $X_1, ..., X_p$ son independientes por lo que la función de densidad f_k se puede representar de la siguiente forma

$$f_k(X) = \prod_{j=1}^{p} f_{jk}(X_j) \tag{3.36}$$

con f_{jk} la función de densidad del predictor X_i dado la clase k, la ventaja de (3.36) es que en lugar de calcular directamente $p_k(x)$, se usa π_k que es relativamente fácil de estimar a partir de la variable de respuesta Y, por otro lado, estimar f_{jk} resulta más complicado a menos que se especifique la forma de estas funciones de densidad. Si no se encuentra la manera de estimar f_{jk} podemos construir el modelo realizando la suposición de que estas funciones de densidad siguen una distribución normal, con esta nueva suposición el modelo se lo denomina clasificador de *Naive Bayes Gaussiano*

$$f_{jk}(x) = \frac{1}{\sqrt{2\pi\sigma_k^2}} exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right) \tag{3.37}$$

con μ_k y σ_k^2 estimados por máxima verosimilitud. De (3.36) y (3.37) se construye

el clasificador de Naive Bayes (3.38), la regla de clasificación es bastante simple ya que para una observación x se la clasifica en la clase k talque que maximice $p_k(x)$. A pesar de las suposiciones un tanto optimistas y su simplicidad, el clasificador de Naive Bayes Gaussiano ofrece buenos resultados y en ciertas ocasiones supera a modelos más sofisticados

$$\begin{aligned} p_k(x) = \frac{\pi_k \prod_{j=1}^{p} f_{jk}(x_j)}{\sum_{i=1}^{K} \pi_i f_i(x)} \rightarrow p_k(x) \propto \pi_k \prod_{j=1}^{p} f_{jk}(x_j) \\ G(x) = \underset{k}{\text{argmax}} \left\{ \pi_k \prod_{j=1}^{p} f_{jk}(x_j) \right\} \end{aligned} \tag{3.38}$$

Naive Bayes como modelo aditivo generalizado

De (3.17) y (3.18) aplicando transformación logit y sea k una clase de la variable respuesta Y, una forma alternativa de escribir el clasificador de Naive Bayes es la siguiente

$$\begin{aligned} log\left(\frac{P(Y=k|X)}{P(Y=l|X)}\right) &= log\left(\frac{\pi_k f_k(X)}{\pi_l f_l(X)}\right) \\ &= log\left(\frac{\pi_k \prod_{j=1}^{p} f_{jk}(X_j)}{\pi_l \prod_{j=1}^{p} f_{jl}(X_j)}\right) \\ &= log\left(\frac{\pi_k}{\pi_l}\right) + \sum_{j=1}^{p} log\left(\frac{f_{jk}(X_j)}{f_{jl}(X_j)}\right) \\ &= \alpha_k + \sum_{j=1}^{p} f_{jk}(X_j) \end{aligned} \tag{3.39}$$

Para tener una visión más clara de este concepto, para un problema de clasificación binario [5] y aplicando el modelo de Naive Bayes se tiene el clasificador como un

[5] Problema de clasificación en que la variable respuesta Y toma solo dos valores

modelo aditivo generalizado.

$$\begin{aligned} log\left(\frac{P(Y=1|X)}{P(Y=0|X)}\right) &= log\left(\frac{P(Y=1|X)}{1-P(Y=1|X)}\right) \\ &= log\left(\frac{\pi_1}{\pi_0}\right) + \sum_{j=1}^{p} log\left(\frac{f_{j1}(X_j)}{f_{j0}(X_j)}\right) \\ g(P(Y=1|X)) &= log\left(\frac{\pi_1}{\pi_0}\right) + \sum_{j=1}^{p} log\left(\frac{f_{j1}(X_j)}{f_{j0}(X_j)}\right) \\ g(P(Y=1|X)) &= \alpha_k + \sum_{j=1}^{p} f_{jk}(X_j) \end{aligned} \tag{3.40}$$

3.3. Evaluación y selección de modelos

Hasta ahora se ha hecho un repaso sobre algunos de los algoritmos de clasificación supervisada utilizados, algunos para resolver problemas de clasificación binaria como las máquinas de soporte vectorial y los árboles boosting, modelos más generales en que la variable respuesta Y tiene k clases como los bosques aleatorios y el clasificador de Naive Bayes.

Generalmente el rendimiento de un modelo de aprendizaje hace referencia a la capacidad de predicción que este tiene cuando se enfrenta a conjuntos de datos que no hayan sido parte del entrenamiento, en la práctica, la evaluación de un modelo es crucial ya que nos da las pautas a seguir para la elección del mejor método de aprendizaje, y nos brinda medidas para cuantificar la calidad de predicción.

En esta sección se describirán los métodos clave para la evaluación y como estos son usados para la elección de un modelo, enfocándose a problemas de clasificación supervisada.

3.3.1. Sesgo, Varianza y Complejidad del modelo

Primero vamos a considerar el caso en que Y esta en una escala cuantitativa, para luego pasar al caso de escala cualitativa. Tenemos la variable de respuesta Y, un vector de predictores X y el modelo de predicción $\hat{f}$ entrenado a partir del conjunto de de vectores de entrenamiento τ. Los costos asociados a que el modelo $\hat{f}$ realice predicciones erróneas se mide con la *función de pérdida* denotada por L, que es el error de medición entre X y $\hat{f}(X)$. Para el caso cuantitativo las elecciones más comunes

de funciones de pérdida son

$$L(Y, \hat{f}(X)) = \begin{cases} (Y - \hat{f}(X))^2 & \text{error cuadrático} \\ |Y - \hat{f}(X)| & \text{error absoluto} \end{cases} \tag{3.41}$$

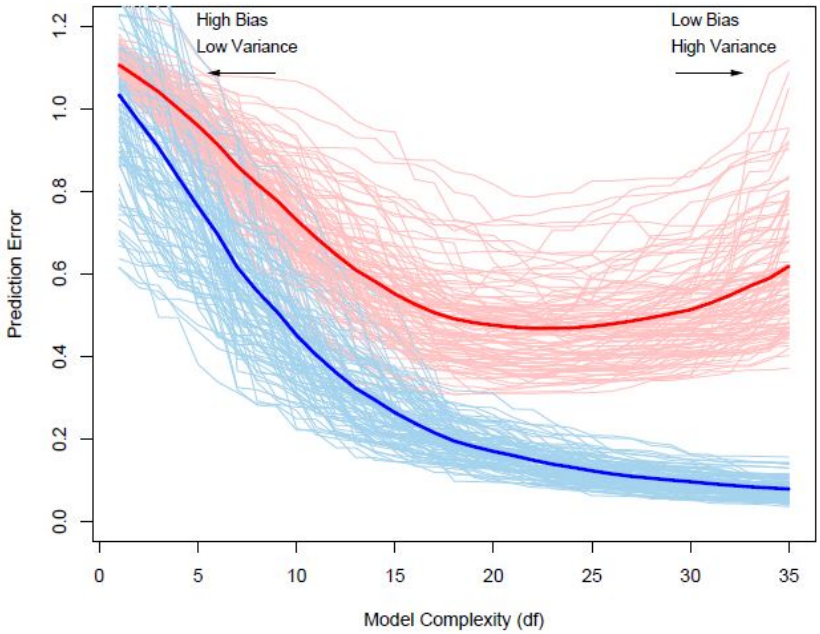

Figura 3.9: Comportamiento del error del conjunto de test y el error del conjunto de entrenamiento como un modelo en función de su complejidad. Las curvas de color azul claro muestran el error de entrenamiento $\overline{err}$, mientras que las curvas de color rojo claro muestran el error de test Err_τ, a medida que aumenta la complejidad del modelo. Las curvas resaltadas muestran el error de test esperado Err y el error de entrenamiento esperado $E[\overline{err}]$ (Hastie, Tibshirani y Friedman, 2009, p.220)

El *error de test* o también denominado *error de generalización*, es el error de predicción del modelo frente a un conjunto de datos independientes, el mejor modelo es aquel que minimice la función de pérdida ya que en general para un conjunto de datos independientes no se conoce a las clases a las que estos pertenecen

$$Err_\tau = E[L(Y, \hat{f}(X)|\tau] \tag{3.42}$$

El conjunto de entrenamiento τ es fijo y el error de test se refiere al error para este conjunto de entrenamiento específico. Una cantidad relacionada es el *error de predicción esperado* o *error de test esperado*

$$Err = E[L(Y, \hat{f}(X))] = E[Err_\tau] \tag{3.43}$$

El *error de entrenamiento* es el promedio de los errores de medición sobre el conjunto de entrenamiento

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) \tag{3.44}$$

A medida que el modelo se convierte más complejo se incrementa el uso del conjunto de entrenamiento, logrando que el modelo se adapte a estructuras escondidas más complejas y producto de esto disminuye el sesgo pero se incrementa la variabilidad, el objetivo es encontrar un modelo equilibrado que nos ofrezca el menor error de test esperado. Podemos apreciar en la **Figura 3.9** que el error de entrenamiento no es un buen estimador del error de test, ya que mientras el modelo es más complejo el error de entrenamiento decrece a valores cercanos a cero, pero el error de test se incrementa

Considerando ahora el caso en que Y esta en una escala cualitativa, es decir que puede tomar K clases distintas etiquetadas por conveniencia $1, 2, ..., K$, generalmente se intenta estimar $p_k(X) = P(G = k|X)$ a través de un clasificador $\hat{G}$. Las funciones de pérdida usuales para medir la calidad de de las predicciones del clasificador son

$$\begin{aligned} L(G, \hat{G}(X)) &= I(G \neq \hat{G}(X)) \quad \text{pérdida 0-1} \\ L(G, \hat{G}(X)) &= exp\{G \cdot \hat{G}(X)\} \quad \text{pérdida exponencial} \\ L(G, \hat{p}(X)) &= -2log(\hat{p}_G(X)) \quad \text{desviación} \end{aligned} \tag{3.45}$$

Es importante tener en mente dos objetivos clave, el primero es la selección del modelo, que se refiere a estimar el desempeño de diferentes modelos y escoger de estos el mejor, el segundo es la evaluación del modelo, una vez seleccionado el modelo idóneo, se busca estimar su error de generalización

Figura 3.10: Representación gráfica de la división del conjunto de datos en sus respectivos tres conjuntos: entrenamiento, validación y test (Hastie, Tibshirani y Friedman, 2009, p.222)

Como se describió en la **sección 3.1.2** el conjunto de datos se divide aleatoriamente en tres partes: entrenamiento, validación y test. El conjunto de validación está destinado para la selección del modelo y el conjunto de test para su evaluación.

No hay una regla para elegir el porcentaje de datos a cada una de las tres partes, pero usualmente se elige valores entre un 50 % a 80 % para el conjunto de entrenamiento y 20 % a 50 % para los conjuntos de validación y test, teniendo en cuenta que los datos de entrenamiento no pueden ser menos de la mitad del conjunto original.

3.3.2. Descomposición sesgo-varianza

El error de predicción esperado utilizando la función de pérdida error cuadrático, nos ofrece información importante para un problema de aprendizaje estadístico, ya que se puede descomponer el error en términos de sesgo y varianza. Asumiendo un problema de regresión en que $Y = f(X) + \epsilon$, donde $E(\epsilon) = 0$ y $Var(\epsilon) = \sigma_\epsilon^2$

$$\begin{gathered} Err(x) = E[(Y - \hat{f}(x))^2 | X = x] = \\ E[(Y - E[\hat{f}(x)] + E[\hat{f}(x)] - \hat{f}(x))^2 | X = x] = \\ E[(y - E[\hat{f}(x)])^2] + E[(\hat{f}(x) - E[\hat{f}(x)])^2] = \\ E[(f(x) + \epsilon - E[\hat{f}(x)])^2] + E[(\hat{f}(x) - E[\hat{f}(x)])^2] = \\ \sigma_\epsilon^2 + (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] = \\ \sigma_\epsilon^2 + sesgo(\hat{f}(x))^2 + var(\hat{f}(x)) \end{gathered} \tag{3.46}$$

El primer término es un error irreducible sin importar que tan bien estimemos $f(x)$, a menos que se tenga $\sigma_\epsilon^2 = 0$. El segundo término es el cuadrado del sesgo, el cual se refiere a que tan alejado está la estimación del valor real de $f(x)$. El último término es la varianza del modelo. Modelos más complejos inducen a reglas de clasificación más complejas lo que podría llevar a un posible sobre ajuste, es decir, se puede intentar modelar a detalle la estructura del conjunto de entrenamiento, pero lo que produciría es que si se realizan pequeños cambios a este conjunto o si se intenta predecir sobre un conjunto de datos independientes, el modelo no logrará identificar estas nuevas estructuras, provocando así una alta variabilidad

3.3.3. Optimismo del error de entrenamiento

En base a las definiciones de la **sección 3.3.1** dado un conjunto de entrenamiento $\tau = \{(x_1, y_1), ..., (x_n, y_n)\}$ el error de generalización o de test para el modelo $\hat{f}$ es

$$Err_\tau = E_{X',Y'}[L(Y', \hat{f}(X'))|\tau] \tag{3.47}$$

con (X', Y') una observación perteneciente al conjunto de test. La media de todos los errores de generalización de los posibles conjuntos de entrenamiento τ es el error de predicción esperado

$$Err = E_\tau[E_{X',Y'}[L(Y', \hat{f}(X'))|\tau]] \tag{3.48}$$

Como el modelo $\hat{f}$ se es ajustado sobre los datos de entrenamiento, el error de entrenamiento $\overline{err} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i))$ generalmente suele ser menor que el error de generalización Err_τ, por lo que tomar $\overline{err}$ como su estimación no es conveniente. Err_τ puede verse como un error de muestra, debido a que los datos en el conjunto de test no coinciden con los de entrenamiento, el *error de muestra* es el siguiente

$$Err_{in} = \frac{1}{N}\sum_{i=1}^{N} E_{Y'}[L(Y_i', \hat{f}(x_i))|\tau] \tag{3.49}$$

donde Y' indica que se observa N nuevas observaciones de la variable de respuesta Y para cada una de las observaciones de entrenamiento x_i, $i = 1, ..., N$. Ahora se define el *optimismo* como

$$op = Err_{in} - \overline{err} \tag{3.50}$$

La media de los optimismos sobre todos los conjuntos de entrenamiento τ denotada por ω es

$$\omega = E_y[op] \tag{3.51}$$

Como las predicciones para x_i, $i = 1, ..., N$ son fijas y esta última expresión depende de los valores de respuesta del conjunto de entrenamiento, en lugar de denotarla por τ se lo hará por y. Podemos estimar solo el error esperado ω en lugar que op de la misma manera que podemos estimar el error esperado Err en lugar del error de generalización Err_τ.

La cantidad en que el error $\overline{err}$ subestima al error Err_τ depende en que tan fuerte es la relación entre y_i y $\hat{f}(x_i)$, $i = 1, ..., N$, es decir, entre más ajustemos el modelo a los datos, el optimismo se incrementará. Una forma de estimar el error de generalización es estimar el optimismo y luego adicionar al error de entrenamiento err.

3.3.4. Validación cruzada

Uno de los métodos más simples y más usados en la práctica para la evaluación de un modelo de aprendizaje $\hat{f}$ es sin duda la validación cruzada, ya que estima directamente el error de predicción esperado Err. Debido a la aleatoriedad de la partición del conjunto de datos en las tres partes (entrenamiento, validación y test), puede darse el caso que las observaciones del conjunto de test puedan ser fácil o difíciles de clasificar o predecir su valor, por lo que intentar estimar Err_τ a partir de una partición predefinida del conjunto de datos no es una buena estrategia.

K-fold validación cruzada

Tener un conjunto de datos considerablemente grande sería lo mejor para poder particionarlo en sus tres partes y tener suficiente información en cada uno de estos, pero en la práctica esto no siempre sucede. Para solucionar este problema el método k-fold validación cruzada divide el conjunto de datos D en K grupos (folds) uniformes $D_1, ..., D_k$. Cada uno de los D_i será tratado como conjunto de validación y usado para predecir el error de predicción, el resto de grupos se usará para el entrenamiento del modelo, es decir, calculamos el error de predicción en el grupo D_k del modelo entrenado sobre $D \setminus \bigcup_{j \neq k} D_j$, repetimos este proceso para cada uno de los K grupos y combinamos sus resultados

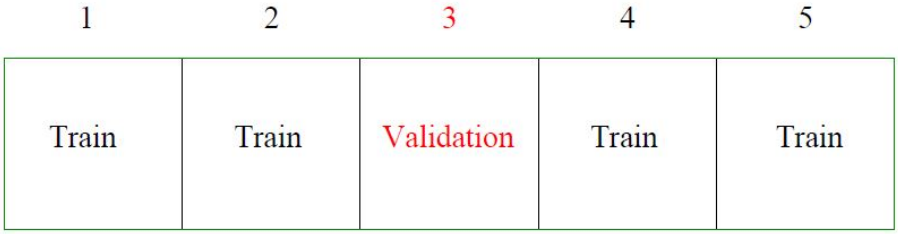

Figura 3.11: Representación gráfica de la división del conjunto de datos para 5-folds validación cruzada (Hastie, Tibshirani y Friedman, 2009, p.242)

Sea $\kappa : \{1, .., N\} \rightarrow \{1, ..., K\}$ una función indexadora que indica a cual de los K grupos pertenece una observación. Denotamos por $\hat{f}^k$ al modelo entrenado con $D \setminus \bigcup_{j \neq k} D_j$. Así el error de predicción estimado por validación cruzada k-fold es

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{\kappa(i)}(x_i)) \tag{3.52}$$

Valores usuales para K utilizados en la práctica es son 5 y 10, un caso especial es cuando $K = N$ comúnmente denominado como *leave-one-out cross-validation*, que

traducido literalmente significa *dejar uno fuera de la validación cruzada*, ya que en este caso $\kappa_i = i$ y para la observación x_i el modelo entrenado es a partir de todos datos a excepción de esa observación. Ahora dado un conjunto de modelos indexados por un parámetro de ajuste α denotados por $\hat{f}(x, \alpha)$ que representa el α-esimo modelo entrenado sin el k-ésimo grupo de datos, así por (3.52) tenemos

$$CV(\hat{f}, \alpha) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{\kappa(i)}(x_i, \alpha)) \tag{3.53}$$

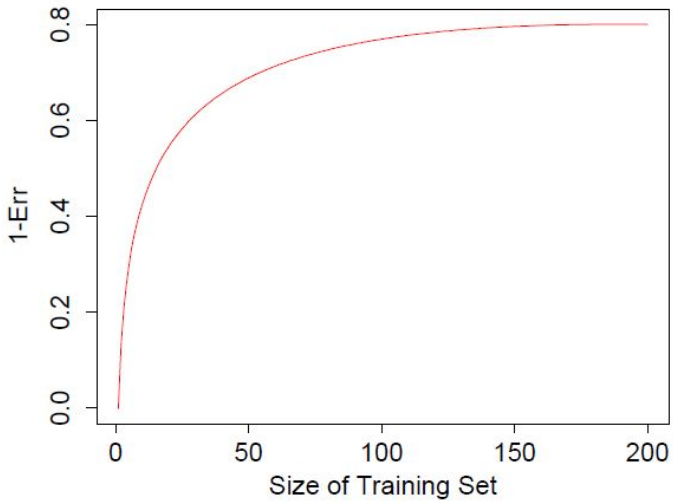

Figura 3.12: Curva de aprendizaje hipotética para un clasificador: gráfico $1 - Err$ versus el tamaño del conjunto de entrenamiento N (Hastie, Tibshirani y Friedman, 2009, p.243)

(3.53) nos provee el error de predicción en función del parámetro de ajuste α, por lo que escogeremos el valor de α tal que minimice $CV(\hat{f}, \alpha)$. Es importante saber que valores para K elegir, para $K = N$ se tiene un error de validación cruzada con muy poco sesgo pero con gran variabilidad y otro aspecto importante es el costo computacional que tomaría entrenar N modelos. Por otro lado, para $K = 5$ o $K = 10$ el error de validación cruzada tendrá menos variabilidad pero el sesgo podría ser un problema dependiendo de cómo varía el rendimiento del modelo de aprendizaje con el tamaño del conjunto de entrenamiento.

3.4. Métricas de evaluación para clasificación

Una vez seleccionado el modelo final que utilizaremos para resolver una tarea en cuestión, repasaremos las principales métricas para cuantificar el desempeño de

Algoritmo 4: K-fold validación cruzada para problema de clasificación

Particionar D en $[D_1, ..., D_K]$ partes iguales
Para $i = 1$ **hasta** K **:**
  $G_i \leftarrow$ entrenar el clasificador sobre $D \setminus D_i$
  $\hat{Err}_i \leftarrow$ evaluar el clasificador G_i sobre $D \setminus D_i$
$\hat{Err} = \frac{1}{K} \sum_{i=1}^{K} \hat{Err}_i$
$\hat{\sigma}^2_{Err} = \frac{1}{K} \sum_{i=1}^{K} (\hat{Err}_i - \hat{Err})^2$
Salida: $(\hat{Err}, \hat{\sigma}^2_{Err})$

este. Las métricas utilizadas dependen del tipo de tarea que se resuelve (clasificación o regresión), en este caso centraremos nuestra atención sobre las métricas para problemas de clasificación y aún de más interés los problemas de clasificación binaria como lo es la detección de noticias falsas. Las métricas que se seguirán en esta sección serán las que recomiendan Zaki y Meira (2014).

3.4.1. Métricas de rendimiento en clasificación

Para medir el rendimiento del modelo utilizaremos el conjunto de test como se repasó en la **sección 3.1.2**. Por T denotaremos al conjunto de test compuesto de n observaciones, $\{c_1, ..., c_K\}$ las K posibles clases que toma la variable de respuesta Y, y_i a la clase a la que pertenece una observación $x_i \in T$ y su predicción hecha por el clasificador G por $\hat{y}_i$

La *tasa de error* es la fracción de predicciones incorrectas hechas por el clasificador G en el conjunto de test T

$$Err = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_i) \tag{3.54}$$

La tasa de error es una estimación de la probabilidad de clasificar erróneamente una observación, mientras menor sea esta tasa mejor es el rendimiento del modelo. La *exactitud* es la fracción de predicciones correctamente hechas por el clasificador G en el conjunto de test, la exactitud y la tasa de error se relacionan de la siguiente forma

$$Acc = \frac{1}{n} \sum_{i=1}^{n} I(y_i = \hat{y}_i) = 1 - Err \tag{3.55}$$

Similar a la tasa de error, la exactitud nos ofrece una estimación de la probabilidad de clasificar correctamente una observación, entre mayor sea esta exactitud

mayor es el rendimiento del modelo. Tanto la tasa de error y la exactitud son medidas globales para el modelo y no consideran explícitamente las clases a las que pertenecen las observaciones, por lo que considerar los aciertos y fallos que realiza el clasificador, dentro de las distintas clases a las que pertenecen las observaciones del conjunto de test, nos proporcionará medidas más informativas.

Denotaremos por $T_1, ..., T_K$ a la partición del conjunto de test basado en la clase a la que pertenecen las observaciones

$$T_j = \{x_i \in T : y_i = c_j\} \tag{3.56}$$

con $n_i = |T_i|$ representando el tamaño de la partición de la clase c_i, $i = 1, ..., K$. Ahora, denotaremos por $R_1, ..., R_K$ a la partición del conjunto de test, pero en este caso, basado en la predicción de clase a la que pertenecen las observaciones

$$R_j = \{x_i \in T : \hat{y}_i = c_j\} \tag{3.57}$$

con $m_i = |R_i|$ representando el tamaño de la partición pero de la clase predicha c_i, $i = 1, ..., K$. Las dos particiones del conjunto de test nos abre la posibilidad de crear una matriz de contingencia N de tamaño $K \times K$, comúnmente llamada *matriz de confusión* definida de la siguiente manera

$$N(i, j) = n_{ij} = |Ri \cap T_j| = |\{x_a \in T : \hat{y_a} = c_i \wedge y_a = c_j\}| \tag{3.58}$$

donde $i \leq i, j \leq K$. Los elementos de la diagonal n_{ii} de la matriz de confusión, representa el número de observaciones en que las predicciones del clasificador coinciden con la clase c_i, por otro lado, los elementos n_{ij}, $i \neq j$ representan el número de observaciones en que las predicciones no coinciden con la clase respectiva

	clase verdadera		
Clase predicha	c_1	...	c_K
c_1	n_{11}	...	n_{1K}
...	...	...	...
c_K	n_{K1}	...	n_{KK}

Tabla 3.1: Matriz de confusión para K clases

La *precisión* del clasificador G para una clase en específico, es la proporción de predicciones correctas dentro de las observaciones que fueron clasificadas en dicha

clase

$$prec_i = \frac{n_{ii}}{m_i} \tag{3.59}$$

con m_i el número de observaciones clasificadas en la clase c_i. La *precisión general* del clasificador G es el promedio ponderado de las precisiones de cada clases.

$$Prec = \sum_{i=1}^{K} \left(\frac{m_i}{n}\right) prec_i = \frac{1}{n} \sum_{i=1}^{k} n_{ii} \tag{3.60}$$

(3.60) es la exactitud (3.55) utilizando la matriz de confusión. La cobertura de G para una clase en específico es la fracción de predicciones correctas dentro de las observaciones que pertenecen a dicha clase

$$rec_i = \frac{n_{ii}}{n_i} \tag{3.61}$$

con n_i el número de observaciones que pertenecen a la clase c_i. Puede darse el caso en que la cobertura de un clasificador sea cercana a uno mientras que se tiene una baja precisión, por otro lado, se podría tener un clasificador con una precisión muy buena pero su cobertura no es la deseada, lo ideal para un modelo es que tanto la precisión como la cobertura tengan valores cercanos a uno. El *F-score* para una clase en específico, es una medida que combina la información de ambas métricas, balancea la precisión y cobertura calculando su media armónica[6]

$$F_i = \frac{2}{\frac{1}{prec_i} + \frac{1}{rec_i}} = 2\frac{prec_i \cdot rec_i}{prec_i + rec_i} = 2\frac{n_{ii}}{n_i + m_i} \tag{3.62}$$

mientras más grande sea el valor de F-score mejor es nuestro clasificador. El *F-score general* es el promedio de los F-score para cada una de las clases

$$F = \frac{1}{K} \sum_{i=1}^{K} F_i \tag{3.63}$$

3.4.2. Clasificación binaria

Los problemas de clasificación binario es de especial interés para este trabajo ya que la detección de noticias falsas puede ser modelizado como un problema de

[6]Dados los valores $x_1, ..., x_n$, su media armónica es $n / \sum_{i=1}^{n} \frac{1}{x_i}$

estos. Consideramos que solo existen dos clases c_1 y c_2, a las cuales llamaremos clase positiva y negativa respectivamente. Con dos clases la matriz de confusión resultante será de tamaño 2×2, y sus elementos se les asigna nombres especiales

	Clase verdadera	
Clase predicha	Positiva(c_1)	Negativa(c_2)
Positiva(c_1)	Verederos positivos (TP)	Falsos positivos (FP)
Negativa(c_2)	Falsos negativos (FN)	Verdaderos negativos (TN)

Tabla 3.2: Matriz de confusión para dos clases

- Verdaderos positivos (TP): Número de observaciones que G clasificó correctamente en la clase positiva

$$TP = n_{11} = |\{x_i : \hat{y}_i = y_i = c_1\}| \tag{3.64}$$

- Falsos positivos (FP): Número de observaciones que G clasificó que pertenecen a la clase positiva pero en realidad pertenecen a la clase negativa

$$FP = n_{12} = |\{x_i : \hat{y}_i = c_1 \wedge y_i = c_2\}| \tag{3.65}$$

- Falsos negativos (FN): Número de observaciones que G clasificó que pertenecen a la clase negativa pero en realidad pertenecen a la clase positiva

$$FN = n_{21} = |\{x_i : \hat{y}_i = c_2 \wedge y_i = c_1\}| \tag{3.66}$$

- Verdaderos negativos (TN): Número de observaciones que G clasificó correctamente en la clase negativa

$$TN = n_{22} = |\{x_i : \hat{y}_i = y_i = c_2\}| \tag{3.67}$$

Reescribiendo las métricas vistas para un problemas de clasificación con $K = 2$ clases, tenemos la exactitud y la tasa de error como

$$\begin{aligned} Err &= \frac{FP + FN}{n} \\ Acc &= \frac{TP + TN}{n} \end{aligned} \tag{3.68}$$

Como vimos anteriormente estas son medidas globales, reescribiendo primero

la precisión para cada una de las dos clases tenemos

$$\begin{aligned} prec_P &= \frac{TP}{m_1} = \frac{TP}{TP+FP} \\ prec_N &= \frac{TN}{m_2} = \frac{TN}{TN+FN} \end{aligned} \tag{3.69}$$

De la misma forma procedemos para las coberturas de la clase positiva y negativa, la *tasa de verdaderos positivos* o comúnmente denominada *sensibilidad* es la fracción de observaciones correctamente clasificadas respecto al total de observaciones de la clase positiva, en cambio, la *tasa de verdaderos negativos* o comúnmente denominada *especificidad* es la fracción de observaciones correctamente clasificadas respecto al total de observaciones de la clase negativa

$$\begin{aligned} TPR = rec_P &= \frac{TP}{n_1} = \frac{TP}{TP+FN} \\ TFR = rec_N &= \frac{TN}{n_2} = \frac{TN}{TN+FP} \end{aligned} \tag{3.70}$$

La *tasa de falsos negativos* y la *tasa de falsos positivos* se relacionan con (3.70) de la siguiente manera

$$\begin{aligned} FNR &= \frac{FN}{n_1} = \frac{FN}{TP+FN} = 1 - TPR \\ FPR &= \frac{FP}{n_2} = \frac{FP}{TN+FP} = 1 - TNR \end{aligned} \tag{3.71}$$

Por último, reescribiendo los F-score para cada una de las dos clases tenemos lo siguiente

$$\begin{aligned} F_P &= \frac{2}{\frac{1}{prec_P} + \frac{1}{rec_P}} = \frac{2TP}{2TP+FP+FN} \\ F_N &= \frac{2}{\frac{1}{prec_N} + \frac{1}{rec_N}} = \frac{2TN}{2TN+FN+FP} \end{aligned} \tag{3.72}$$

3.4.3. Análisis ROC

Cuando existen dos clases, uno de los métodos más populares utilizados para evaluar el rendimiento de un clasificador es el de *características de funcionamiento del receptor* (ROC). Para el análisis ROC necesitaremos que nuestro clasificador G retorne *puntuaciones* de la clase positiva de cada uno de las observaciones del conjunto

de test para poder ordenarlos en forma descendente. Por ejemplo, se puede utilizar como puntuación la probabilidad a posteriori $P(Y = c_1|X = x_i)$ obtenida del clasificador G. En general para un problema de clasificación binaria se utiliza un umbral ρ para el cual se clasifican las observaciones con calificaciones sobre este en la clase positiva y los restantes en la case negativa. El análisis ROC se basa en graficar el rendimiento del clasificador sobre todos los valores posibles del umbral ρ, es decir, para cada valor de ρ, se grafica la tasa de falsos positivos (FPR) en el eje x y la tasa de verdaderos positivos (TPR) en el eje y, la gráfica resultante se llama denomina *Curva ROC* para el clasificador G.

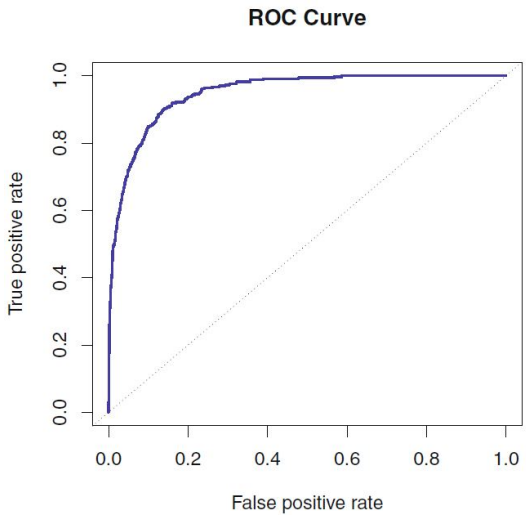

Figura 3.13: El análisis ROC grafica dos tipos de tasas de error a medida que variamos el valor de umbral para la probabilidad a posteriori de las observaciones. La curva ROC ideal es cercana a la esquina superior izquierda, lo que indica una alta tasa de verdaderos positivos y una baja tasa de falsos positivos. La línea punteada representa el clasificador aleatorio (James, Witten, Hastie y Tibshirani, 2013, p. 148)

Denotaremos por $S(x_i)$ a la puntuación de la predicción para la clase positiva de la observación x_i hecha por el clasificador G. Así, definimos los umbrales máximo y mínimo del conjunto de test respectivamente como

$$\rho_{min} = \min_i\{S(x_i)\} \quad \rho_{max} = \max_i\{S(x_i)\} \tag{3.73}$$

Tomando el umbral máximo, iniciamos clasificando todas las observaciones del conjunto de test en la clase negativa, ya que ninguna de estás tendrá una calificación mayor a ρ_{max}, producto de esto, las tasas $TPR = FPR = 0$, que corresponde al punto (0,0) en la **Figura 3.14**. Luego para cada uno de los umbrales hallamos el conjunto

de observaciones clasificadas en la clase positiva

$$R_P(\rho) = \{x_i \in T : S(x_i) > \rho\} \tag{3.74}$$

calculando las respectivas tasas TPR y FPR para obtener el siguiente punto de la curva ROC. Como paso final se toma el umbral ρ_{min}, así, todas las observaciones del conjunto de test serán clasificadas como positivas provocando que $TPR = FPR = 1$ correspondiendo al punto (1,1) en la **Figura 3.13**. La curva ROC nos indica hasta qué punto el clasificador G predice la clase de las observaciones en la positivas por encima de las que predice en la clase negativa

Area bajo la curva ROC

El área bajo la curva ROC, abreviada AUC, puede usarse como medida del desempeño del clasificador G. Como el área total del rectángulo $[0,1] \times [0,1]$, el valor de AUC toma valores dentro de $[0,1]$, cuanto más cercano a uno, mejor es nuestro clasificador. El valor AUC puede ser vista como la probabilidad de que el clasificador G pueda distinguir la clase positiva de la negativa.

Clasificador aleatorio

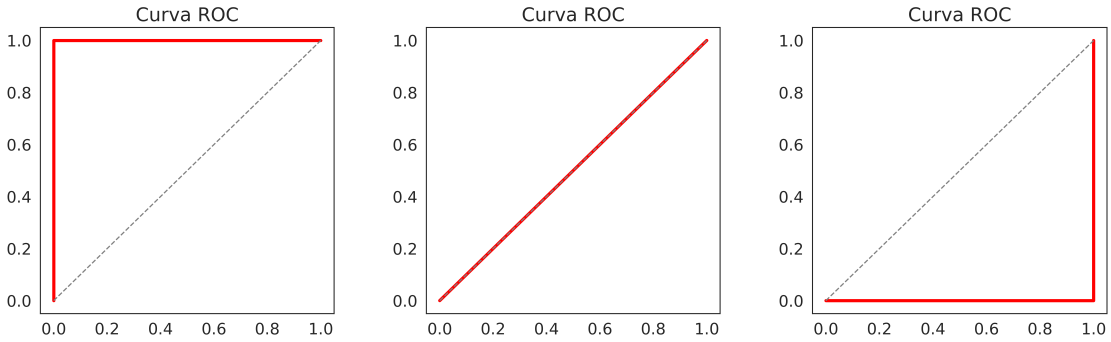

Figura 3.14: Comparación de los tres casos hipotéticos extremos de las curvas ROC. Izquierda: clasificador perfecto. Centro: clasificador aleatorio. Derecha: peor de todos los clasificadores

Es importante notar que un clasificador que realiza sus predicciones aleatoriamente, su curva ROC corresponde a una línea diagonal en el la **Figura 3.14**. Para verlo mejor, consideremos un clasificador que adivina aleatoriamente la clase de una observación como positiva y negativa la mitad de las veces, similar al lanzamiento de una moneda. Esperaríamos que para cada uno de los umbrales ρ las tasas

TPR y FPR siempre sean las mismas, obteniéndose así la curva diagonal. Así, si la curva ROC de cualquier clasificador está por debajo de la diagonal, indica un rendimiento peor que un modelo aleatorio, ya que el area bajo su curva ROC será menor que la del modelo aleatorio, es decir, si algún clasificador tiene un valor AUC menor que 0.5, eso indica un muy mal desempeño, incluso peor que adivinar.

Capítulo 4

Creación, Exploración y Procesamiento de datos

En este capítulo se detalla todo lo relacionado al conjunto de datos utilizado para el modelo de detección de noticias falsas. Se empieza por la explicación del proceso de creación del corpus de noticias, en especial, las fuentes de información de donde se recolectaron el tipo especial de noticias falsas utilizadas en este trabajo. Posterior a esto, se especifica como se realizó el procesamiento de los textos del corpus y las técnicas que se emplearon para esta tarea. Finalmente, se explica los métodos utilizados para la extracción de características lingüísticas de las noticias, con el objetivo de crear el conjunto de datos destinado al entrenamiento de los algoritmos

4.1. Procesamiento del lenguaje natural

Acorde con Vajjala et. al (2020) el procesamiento del lenguaje natural con siglas en inglés (NLP) es un campo que combina la ciencia computacional, la inteligencia artificial y la lingüística, enfocándose en la creación de sistemas capaces de procesar y comprender el lenguaje humano, teniendo aplicaciones dentro de la salud, finanzas, leyes, marketing, recursos humanos y muchas más.

Como se analizó en la **Sección 3.1** el aprendizaje estadístico supervisado tiene un sin número de aplicaciones en la vida real y una de ellas es en resolver una tarea de NLP. Existen muchas tareas que se pueden realizar con NLP, para la realización de este trabajo fueron de interés dos en específico: clasificación de texto, ya que se necesita diferenciar noticias falsas de noticias reales y esto se lo realizará mediante algoritmos de clasificación supervisada vistos en la **Sección 3.2**, y recuperación de

información a partir de los textos como se vio en la sección **Sección 2.1.3**, esto nos proveerá los datos y herramientas necesarias para entrenar los modelos

4.2. Creación del corpus de noticias

Dado la amplia diversidad de formas en que se habla el español en américa latina, el estudio se enfocó en el Ecuador, es decir, el modelo de detección automática desarrollado está centrado en la detección de noticias falsas de tipo sátira política ecuatoriana en redes sociales, específicamente en Facebook. En esta sección se explica como se realizó el proceso de recolección de noticias, las fuentes de donde se las extrajo, el tipo de noticias falsas y un análisis del corpus resultante

4.2.1. Fuentes de información

El corpus consta de una colección de noticias del ámbito ecuatoriano extraídas de distintas páginas dentro de internet: sitios web de principales periódicos nacionales reales y páginas de sátira de corte político en redes sociales, recolectadas en las fechas de septiembre a noviembre del 2020. Las páginas de periódicos nacionales elegidas fueron: *El Comercio* y *El Universo* destinadas a la fracción de noticias reales y las páginas de sátira de corte político fueron: *El Uninverso*, *El Culimercio* y *El Mercioco* destinado a la fracción de noticias falsas

Página	Sitio	Tipo de noticias
El Universo	www.eluniverso.com	Periódico nacional
El Comercio	www.elcomercio.com	Periódico nacional
El Uninverso	www.facebook.com/DiarioUninverso	Sátira Política
El Culimercio	www.facebook.com/elculimercioec	Sátira Política
El Mercioco	www.facebook.com/mercioco	Sátira Política

Tabla 4.1: Páginas de periódicos nacionales y páginas de sátira política utilizadas

Analizando los títulos de las páginas que producen noticias falsas, se puede apreciar que son bastantes similares a los títulos de las páginas de los periódicos, esto es por que los autores de este tipo de noticas intentan imitar el contenido de noticias reales (Lazer et al., 2018). Páginas de este estilo contienen noticias que no son del todo ciertas ya que mezclan hechos reales con información falsa con el objetivo de confundir a los lectores (Posadas et al., 2019)

4.2.2. Extracción de noticias

Las publicaciones que realizan las páginas de sátira política en facebook utilizadas en este trabajo, se caracterizan por ser mensajes cortos, por lo que la información que se obtuvo de las páginas de los periódicos no fue la nota completa, fue el resumen de la noticia proporcionada por los mismos periódicos con el fin de que mantenga el patrón que siguen las páginas de sátira. El proceso de recolección del texto de las páginas en facebook se lo realizó manualmente copiando el contenido, por otro lado, los textos recolectados de las páginas de los periódicos se lo realizó mediante la ayuda de herramientas de *webscrapping* [1].

Figura 4.1: Ejemplo de textos de noticias recolectados para cada una de los tipos de páginas. Izquierda: página de *El Comercio*. Derecha: Página de *El Uninverso*

El conjunto de noticias resultante fue almacenado en un archivo de *valores separados por comas* (csv) con codificación *utf-8* [2] para poder almacenar palabras en español sin que se pierdan las tildes y la letra *ñ*. Finalmenente, para etiquetar las noticias recolectadas, se consideró lo siguientes aspectos

- Las noticias fueron etiquetadas como reales si hay evidencia de que la publicación proviene de una página confiable, como lo es para las páginas web de los periódicos
- Las noticias fueron etiquetadas como falsas si no se encontró evidencia sobre la veracidad de la información presentada y la credibilidad de la página de donde provienen estas noticias, para este caso las páginas de sátira política

[1] Es el proceso de recopilar datos web estructurados de forma automatizada
[2] Tipo de formato de codificación de caracteres

4.2.3. Análisis exploratorio

El objetivo de la detección de noticias falsas es diferenciar las noticias reales de las falsas, esto se lo logrará mediante la clasificación supervisada de las mismas utilizando los algoritmos vistos en la **sección 3.2**. Así, definimos la variable respuesta Y como una variable binaria: esta tomará el valor de 1 si se trata de una noticia falsa y 0 si es una noticias real

$$y_i = \begin{cases} 1 & \text{si la noticia } i \text{ es falsa} \\ 0 & \text{caso contrario} \end{cases} \tag{4.1}$$

El corpus de noticias consta de un total de 1629 noticias, aproximadamente el 59 % de estas pertenecen a El Comercio, ya que debido a la estructura de su página web, facilitó el proceso de recolección de las noticias mediante webscrapping, por lo que se pudo extraer gran cantidad de noticias con una gran velocidad. En general las páginas que crean información falsa tienden a tener un tiempo de vida corto (Allcott y Gentzkow, 2017), por lo que la información que se obtuvo de dos de las páginas de sátira, es mucho menor a la que se obtuvo de las páginas de periódicos nacionales, a excepción de la página El Uninverso que es una página bastante activa que ha publicado bastante contenido desde su creación; la ventaja de esto fue que el tiempo invertido para la extracción de los textos falsos no fuera excesivo y fue viable ejecutarlo manualmente

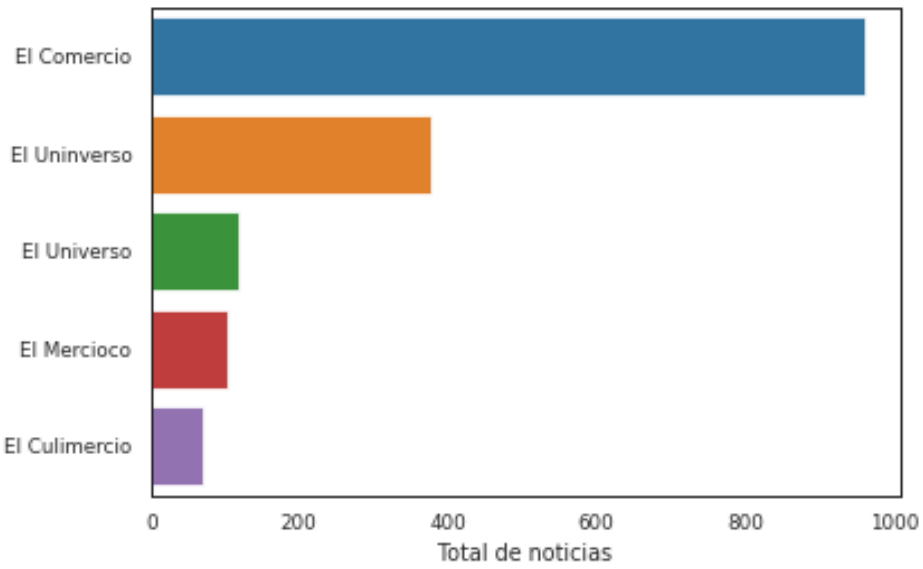

Figura 4.2: Distribución de las noticias recolectadas según la página de donde provienen

Del número total de noticias tenemos que el 34 % corresponden a noticias falsas y el 66 % restante a noticias reales, motivo por el cual se procedió a balancear los datos con una técnica llamada SMOTE la que será detallada posteriormente.

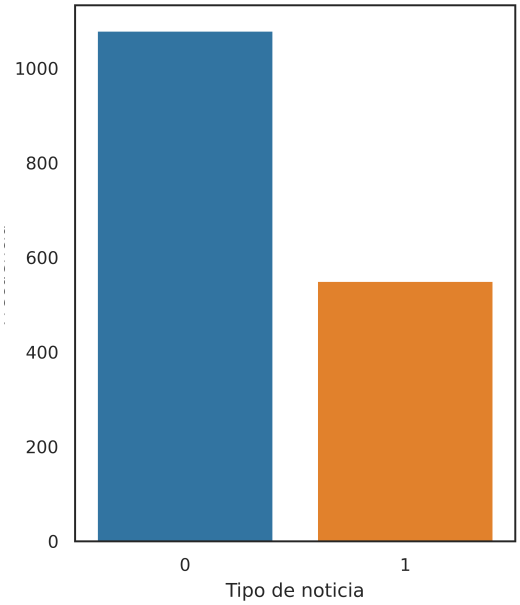

Figura 4.3: Distribución de las noticias recolectadas según el tipo: 1 si es falsa y 0 caso contrario

Analizando las noticias reales y las noticias falsas desde un punto de vista a nivel de términos, podemos apreciar en la **Figura 4.4** que la distribución del número de términos es bastante parecido para ambos grupos, tanto así, que el promedio de términos para las noticias reales es aproximadamente 41 términos mientras que para las noticias falsas es aproximadamente 42. Por otro lado, el número de términos presentes en las noticias falsas son un poco más variables que las reales, teniendo 15.7 y 19.6 las desviaciones aproximadas respectivamente.

4.3. Procesamiento de texto

Antes de extraer las características lingüísticas de las noticias para elaborar la base de datos destinada al proceso de entrenamiento, se realizó un procesamiento previo de los textos de las noticias. En esta sección se detalla dicho proceso, en el que se removió palabras repetitivas que no aportan información alguna sobre las noticias, además se eliminó caracteres, puntuaciones, entre otros elementos no ne-

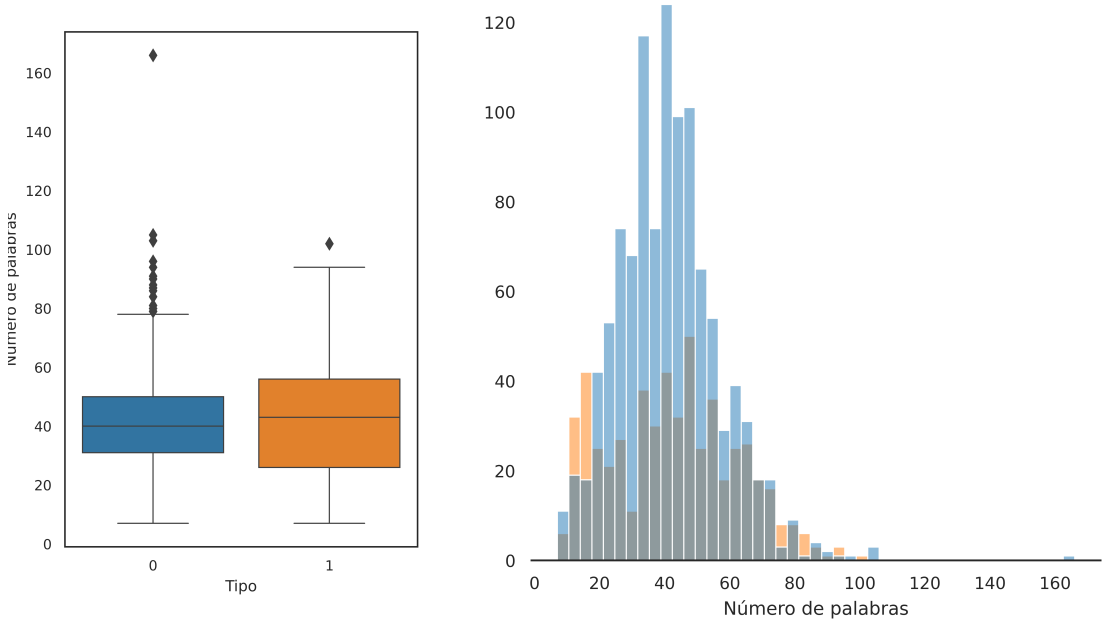

Figura 4.4: Distribución del número de términos para los dos tipos de noticias

cesarios, y finalmente con la ayuda del paquete *spaCy* (Honnibal y Montani, 2017) se transformó todas las palabras a su forma base.

4.3.1. Limpieza de textos

Los componentes de interés en este trabajo son las palabras de cada una de las noticias, motivo por el cual fueron extraídos elementos que no se los puede considerar palabras como tal. Entre los elementos removidos de los textos tenemos: links referentes a páginas externas, etiquetas hacia otras cuentas de usuarios para el caso de noticias de páginas de facebook, hashtags, signos de puntuación, caracteres numéricos, caracteres especiales y se transformó todas las palabras a minúsculas.

4.3.2. Remoción de stop words

Al término *stop words* que su traducción al español sería *palabras vacías*, y como su nombre lo indica, hace referencia a las palabras frecuentemente utilizadas en los idiomas que no son de ayuda por si mismas en la compresión del contexto de la noticia y en general de un texto por lo que usualmente son removidas. Por ejemplo en el idioma español tenemos las palabras: el, la, un, una, unos, de, en, etc.

4.3.3. Lematización

La lematización es el proceso en que dados todas las diferentes formas flexionadas de una palabra se halla su forma base o lema (Vajjala et. al), esto nos ayuda a reducir considerablemente el número de palabras con significados similares de un texto, ya que por ejemplo, si tenemos las palabras: *estudié*, *estudio*, *estudiaba*, aplicando el proceso de lematización el resultado es *estudiar* ya que este es el lema de las palabras anteriores y en consecuencia no se tendrá tres palabras diferentes sino una sola. Esta tarea se la realizó mediante la ayuda de la librería de procesamiento de texto *spaCy* (Honnibal y Montani, 2017), en ella existen una gran variedad de modelos pre-entrenados basados en redes neuronales en distintos idiomas, incluyendo el español. El proceso completo se muestra a continuación.

$$
\begin{gathered}
\textit{Texto DEMOSTRATIVO de procesar los textos procesados:} \\
\textit{Limpieza, sin stop words, y Lematización. 2020! \#nlp} \\
\Downarrow \\
\textit{texto demostrativo de procesar los textos procesados limpieza} \\
\textit{sin stop words lematización} \\
\Downarrow \\
\textit{texto demostrativo procesar textos procesados limpieza} \\
\textit{stop words lematización} \\
\Downarrow \\
\textit{texto demostrativo procesar texto procesar limpieza} \\
\textit{stop words lematización}
\end{gathered}
\tag{4.2}
$$

Veremos en la **Sección 4.4.1** que la técnica utilizada para la extracción de características se ve influenciada por el número de palabras que se tiene en cada uno de los textos, motivo por el cual se analizará el impacto que tiene la remoción de stop words y la lematización al momento de representar numéricamente las noticias en vectores, ya que como vimos, estas dos técnicas nos ayudan a reducir el número de palabras en un texto.

4.4. Representación de texto

Como se analizó en la **Sección 2.1.3** existe una gran variedad de características que se pueden extraer de las noticias, para la realización de este trabajo fueron elegidas características basadas en el contenido de tipo lingüísticas a un nivel de palabra para poder representar de una forma adecuada el vocabulario que caracteriza la forma de hablar de los ecuatorianos. El objetivo de esta sección es encontrar una representación numérica de un texto, para lo que se analizara una de las clásicas técnicas para la representación numérica: frecuencia de término inversa, la cual fue elegida para crear los vectores de la base de datos destinada al entrenamiento del modelo

4.4.1. Frecuencia de término inversa

Acorde con Vajjala et al. (2020), a partir de un vocabulario de palabras V creado del corpus, se asigna a cada palabra de este una única identificación, sin importar el orden ya que esto no afectará al resultado. La idea básica es representar cada una de las noticias como un vector V-dimensional, en el que el valor de cada una de las entradas depende de la técnica de representación que se use, la frecuencia de término inversa, tiene como objetivo cuantificar la importancia de una palabra dada en relación con otras palabras del documento y en el corpus. Si una palabra aparece muchas veces en un documento d pero no ocurre mucho en el resto de los documentos del corpus, entonces la palabra p debe ser de gran importancia para el documento d. Por otro lado, la importancia de p debe aumentar en proporción a su frecuencia de aparición en el documento d, pero al mismo tiempo, su importancia debe disminuir en proporción a la frecuencia de la palabra en otros documentos d del corpus. Matemáticamente, esto se captura usando dos cantidades: TF e IDF que multiplicadas se define $TF\text{-}IDF$.

Sea el corpus de documentos $D = \{d_1, ..., d_M\}$ y el vocabulario $V = \{p_1, ..., p_N\}$, la frecuencia de termino (TF) mide la frecuencia con la que un término aparece en un documento dado. Algo importante a considerar es dado que el número de términos varia en cada documento, un término puede aparecer con mayor frecuencia en un documento más largo en comparación a uno más corto, por lo que se divide el número de ocurrencias del término para la longitud de del documento. La cantidad

TF se define de la siguiente forma

$$TF(p_i, d_j) = \frac{f(p_i, d_j)}{|d_j|} \tag{4.3}$$

donde $f(p_i, d_j)$ es la frecuencia del término p_i del vocabulario V en el documento d_j del corpus D, para $i = 1, ..., N$ y $j = 1, ..., M$. La frecuencia inversa de documentos (IDF) mide la importancia de un término dentro del corpus, pondera con menor peso a los términos que son muy comunes dentro del corpus mientras que, pondera con mayor peso a los términos poco usuales. Como se mencionó en la **Sección 4.3.2**, las stop words no son importantes para la extracción de información, esto se ve reflejado por el IDF de estas palabras, y es debido a que ocurren con frecuencia dentro de la mayoría de los textos del corpus. La cantidad IDF se define de la siguiente forma

$$IDF(p_i) = log\left(\frac{M}{|\{d \in D : p_i \in d\}|}\right) \tag{4.4}$$

donde $|\{d \in D : p_i \in d\}|$ representa el número de textos en los que aparece el término p_i para $i = 1, ..., N$, así la frecuencia de término inversa se define como

$$TF\text{-}IDF(p_i, d_j) = TF(p_i, d_j) \cdot IDF(p_i) \tag{4.5}$$

Para el cálculo de TF-IDF sobre el conjunto de noticias se utilizó la librería de Python *scikit-learn* (Pedregosa et. al, 2011). Aplicando lo anterior al corpus de noticias el vocabulario V resultante consta de 9953 palabras, por lo que el vector que representa a cada uno de los textos del corpus tiene este número de entradas.

La técnica TF-IDF depende del número total de palabras en un corpus, si el vocabulario resultante consta de un excesivo número de palabras, esta cantidad será el número de variables al representar numéricamente los textos, por lo que remover stop words y lematizar las palabras nos ayudarán a reducir el número de variables sin perder la estructura del texto. La base de datos original creada tiene 1629 registros con 9953 variables, pero aplicando las técnicas de reducción mencionadas, el número de variables disminuye a 6336. Analizaremos si reducir en número de variables con estas técnicas afecta en el rendimiento de los modelos de clasificación, por lo que estos dos conjuntos de datos se utilizaron para entrenar los algoritmos y los resultados son presentados en el **Capítulo 5**

Capítulo 5

Modelo automático de identificación

Una vez listos los dos conjuntos de datos que representan numéricamente las noticias del corpus, en este capítulo se hace uso de los mismos para entrenar cada uno de los algoritmos: máquinas de soporte vectorial, bosques aleatorios, árboles boosting y el clasificador de Naive Bayes. Ya que solo se tiene un 34 % de noticias falsas, la base de datos fue balanceada con el objetivo de mejorar la calidad de las predicciones. Al final de esta sección se estudia las diferencias entre los modelos entrenados sobre las dos variantes de conjuntos de datos propuestas y se escogerá el que ofrezca mejores resultados.

5.1. Balanceo de la base de datos

Para balancear los conjuntos de datos se utilizó una técnica denominada *SMOTE* (Bowyer et al., 2011), la cual a diferencia de las tradicionales basadas en el remuestro con reposición, esta crea observaciones sintéticas de la clase minoritaria utilizando los K vecinos más cercanos. La clase minoritaria se sobremuestrea tomando cada una de las observaciones e introduciendo observaciones sintéticas a lo largo de los segmentos que las unen con uno o todos los K vecinos más cercanos, es decir, se toma la diferencia entre una observación y su vecino más cercano y se multiplica esta diferencia por un número aleatorio entre 0 y 1, con lo que se selecciona un punto aleatorio a lo largo entre las dos observaciones. Para x_i una observación se tiene.

$$x^* = x_i + \lambda \cdot (x_i - x_{ki}) \tag{5.1}$$

con x^* la observación sintética generada, x_{ki} el k vecino más cercano de la observación x_i y $\lambda \in [0, 1]$ un número aleatorio. Dependiendo del número de muestras sintéticas requeridas, los K vecinos más cercanos son elegidos aleatoriamente, Bowyer et al. (2011) utilizan 5 vecinos más cercanos por lo que se procederá de la misma forma.

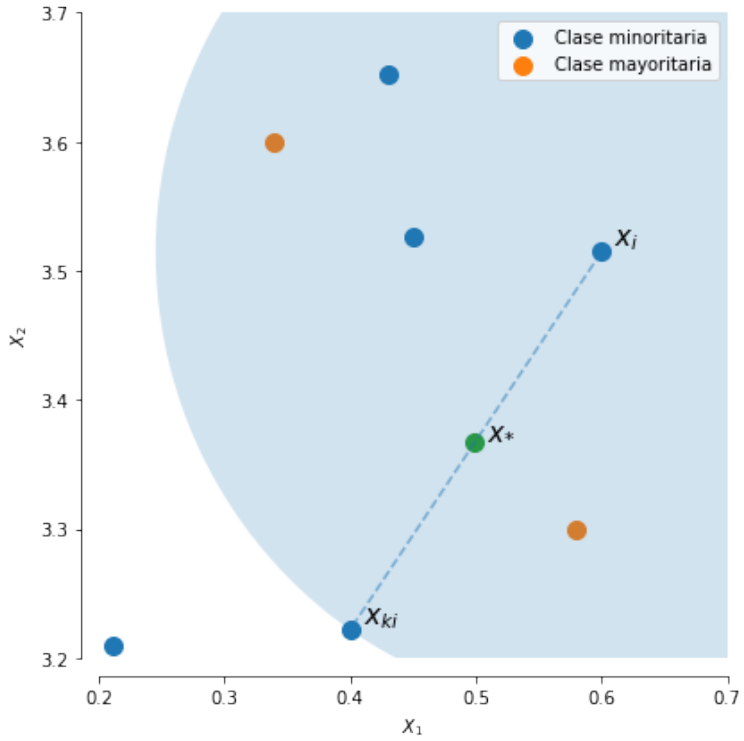

Figura 5.1: Observación sintética generada aleatoriamente a partir de la observación x_i con su K vecino más cercano x_{ki} para un caso de dos variables (Lemaître, 2017)

Para utilizar esta técnica, se eligó la librería de Python *imblearn* (Lemaître, Nogueira y Aridas, 2017) en la que esta implementada la técnica SMOTE. El porcentaje de datos destinado al entrenamiento de los modelos fue del 80 %, mientras que para los conjuntos validación y test, fue del 20 % restante. Se realizó el proceso de balanceo en el conjunto de entrenamiento, agregando observaciones sintéticas de tal forma que el número de noticias falsas sea el mismo que de noticias reales, con el objetivo de que el algoritmo tenga suficientes noticias falsas para identificar los patrones que diferencian a estas de las noticias reales, y para que las predicciones hechas sobre los conjuntos de validación y test, el modelo lo haga sobre datos no sintéticos.

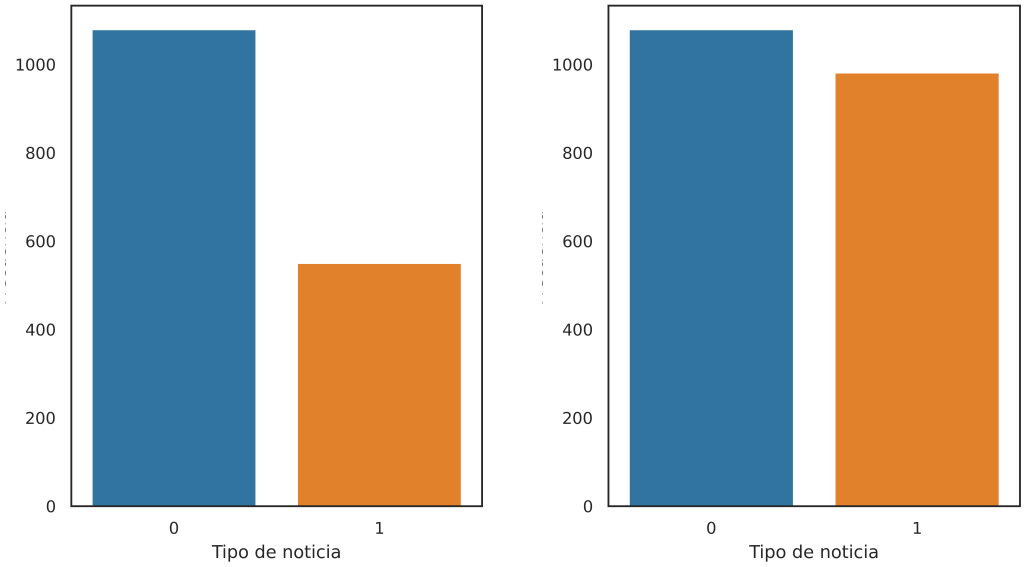

Figura 5.2: Izquierda: Base de datos sin balancear. Derecha: Base de datos aplicado la técnica SMOTE en el conjunto de entrenamiento

5.2. Entrenamiento

Se presenta el proceso de entrenamiento de los algoritmos de los cuatro algoritmos de clasificación supervisada planteados en este trabajo. Los mejores parámetros para los modelos, como se analizó en la **Sección 3.1.2**, fueron escogidos en relación a los que ofrezcan el menor valor de la función de pérdida 0-1 sobre el conjunto de validación y al final se evaluó los modelos finales con las métricas presentadas en la **Sección 3.4.1**, los valores de cada una de las métricas para todos los modelos se encuentran tabulados en la **Tabla A.2**. Cada uno de los algoritmos se encuentran implementados en la librería de Python *scikit-learn* (Pedregosa et. al, 2011).

5.2.1. Modelo 1 (SVM)

Se entrenaron tres modelos de máquinas de soporte vectorial con tres kernels diferentes: lineal, sigmoide y gaussiano. El parámetro de ajuste C, que controla que tan restrictivo es el algoritmo a las observaciones mal clasificadas, fue optimizado del conjunto de valores $C = \{0{,}01, 0{,}5, 1, 5, 25, 50, 100\}$. Los parámetros γ para el kernel gaussiano y α, β para el kernel sigmoide, se tomaron los que se utilizan en la librería *scikit-learn*, $\frac{1}{M \cdot var(X)}$ (M el número de variables y $var(X)$ la varianza de todo el conjunto de datos) el valor para γ y α, y β toma el valor de 0.

Hiperparámetro C óptimo sin stop words y textos lematizados

Para los tres casos un valor muy restrictivo $C = 0{,}01$ retorna mucha más pérdida a comparación al resto, los valores óptimos del parámetro C que ofrecen la menor pérdida para los modelos con kernel sigmoide y lineal es $C = 0{,}5$, por otro lado para el modelo con kernel gaussiano se tiene cinco valores de C que ofrecen la menor pérdida, por lo que se tomó el menor de ellos $C = 1$.

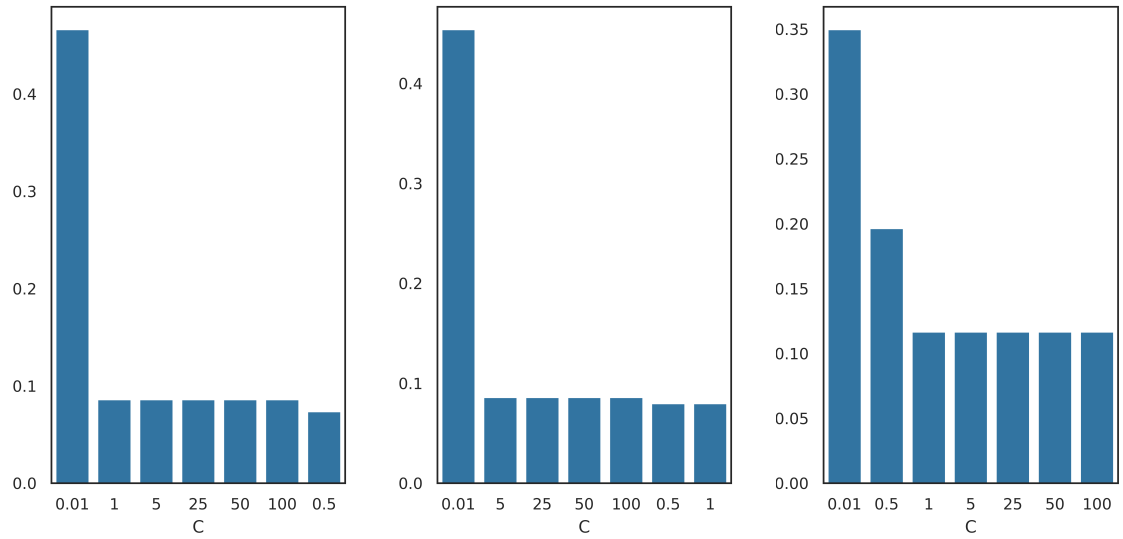

Figura 5.3: Valor de la función de pérdida sobre el conjunto de validación para los distintos valores del parámetro C sin stop words y textos lematizados. Izquierda: kernel lineal. Derecha: kernel gaussiano. Centro: kernel sigmoide

Resultados en el conjunto de test

Se realizaron las predicciones sobre el conjunto de test que consta de 163 noticias reales, con los tres modelos finales escogidos. Analizando las matrices de confusión resultantes se tiene que los modelos con kernel lineal y sigmoide tienen comportamientos muy similares clasificando la mayor parte de las noticias correctamente, en cambio, el modelo con kernel gaussiano clasifica parte de las noticias que son falsas como verdaderas, pero clasifica correctamente las que son noticias reales. Los tres modelos tienden a cometer mas falsos negativos que falsos positivos.

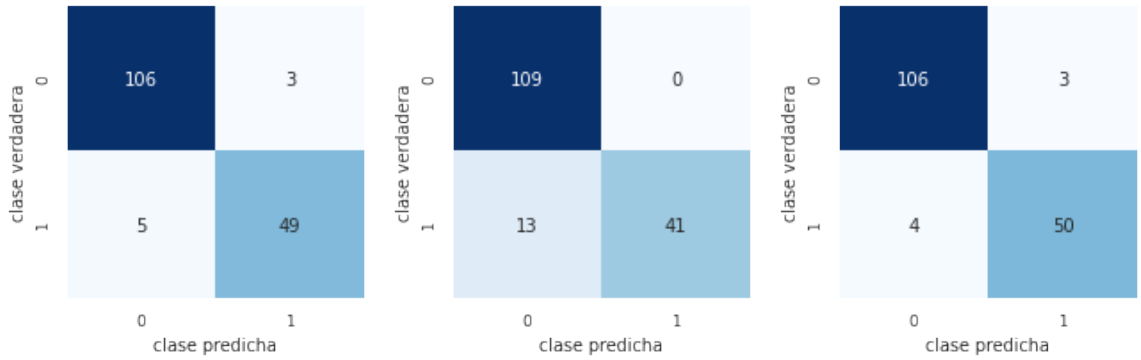

Figura 5.4: Matrices de confusión elaboradas con las predicciones del conjunto de test sin stop words y textos lematizados. Izquierda: kernel lineal. Derecha: kernel sigmoide. Centro: kernel gaussiano

Análisis ROC

Para el cálculo de la curva ROC se utilizó las predicciones de la probabilidad de ser una noticia falsa sobre el conjunto de test, así, los tres modelos presentan una área bajo la curva ROC excelente muy cerca a 1, específicamente se tiene los siguientes valores aproximados expresados en porcentajes: 99,2 %, 99,31 % y 99,24 % para los kernel sigmoide, rbf y lineal respectivamente. Los umbrales para los que se tiene la mayor diferencia entre la tasa de falsos positivos y la tasa de verdaderos positivos son aproximadamente expresados en porcentaje: 28,88 %, 24,05 % y 24,14 %.

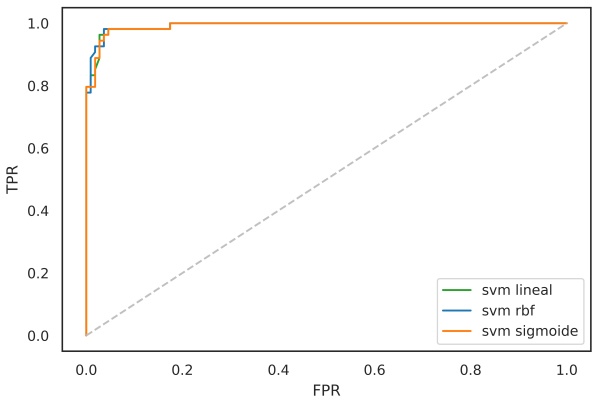

Figura 5.5: Curvas ROC de los tres modelos de máquinas de soporte vectorial sin stop words y textos lematizados

Hiperparámetro C óptimo con stop words y textos no lematizados

De igual forma, para los tres casos un valor muy restrictivo $C = 0{,}01$ retorna mucha más pérdida a comparación del resto, los valores óptimos del parámetro C

son varios para cada uno de los modelos, por lo que se tomó el menor de ellos, teniendo así los siguientes valores: $C = 5$ para el kernel lineal y kernel gaussiano, y $C = 1$ para el kernel sigmoide.

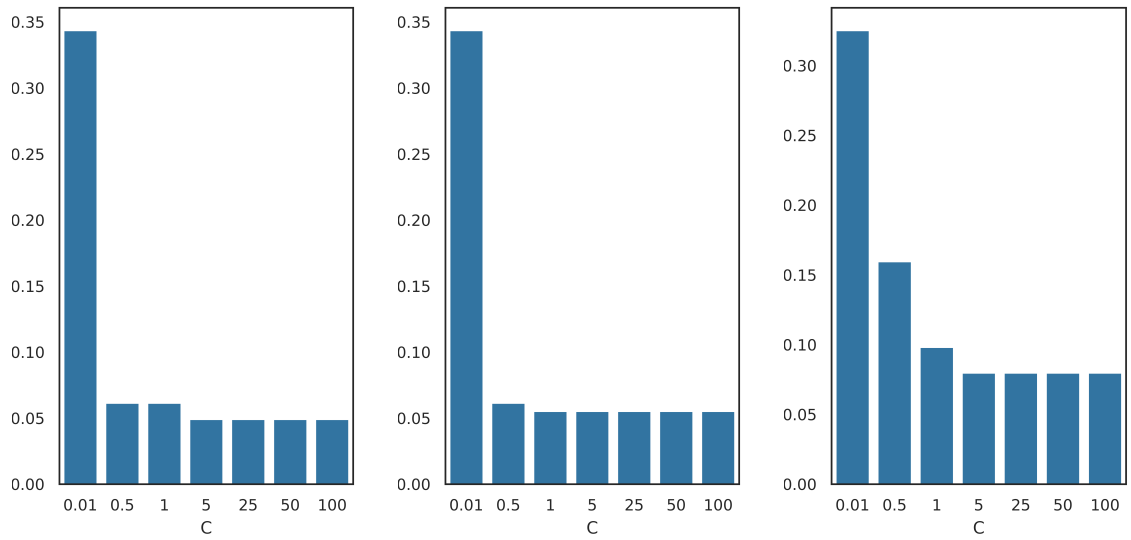

Figura 5.6: Valor de la función de pérdida sobre el conjunto de validación para los distintos valores del parámetro C con stop words y textos no lematizados. Izquierda: kernel lineal. Derecha: kernel gaussiano. Inferior: kernel sigmoide

Resultados en el conjunto de test

Se realizaron las predicciones sobre el conjunto de test, con los tres modelos finales escogidos. Analizando las matrices de confusión resultantes se tiene que los tres modelos tienen comportamientos muy similares clasificando la mayor parte de las noticias correctamente y con una tendencia a cometer mas falsos negativos que falsos positivos, en especial el modelo con kernel gaussiano.

Análisis ROC

Los tres modelos presentan una área bajo la curva ROC excelente muy cerca a 1, específicamente se tiene los siguientes valores aproximados expresados en porcentajes: 99,3 %, 99,2 % y 99,4 % para los kernel sigmoide, rbf y lineal respectivamente. Los umbrales para los que se tiene la mayor diferencia entre la tasa de falsos positivos y la tasa de verdaderos positivos son aproximadamente expresados en porcentaje: 16,58 %, 35,51 % y 15,37 %.

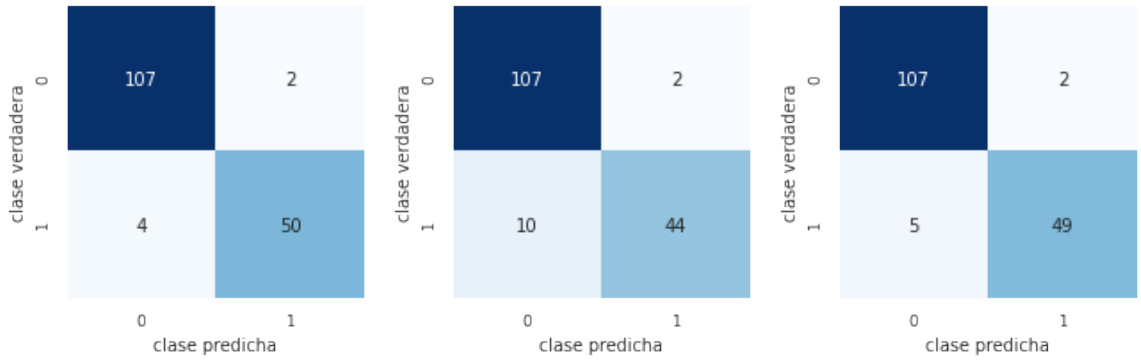

Figura 5.7: Matrices de confusión elaboradas con las predicciones del conjunto de test con stop words y textos no lematizados. Izquierda: kernel lineal. Derecha: kernel sigmoide. Centro: kernel gaussiano

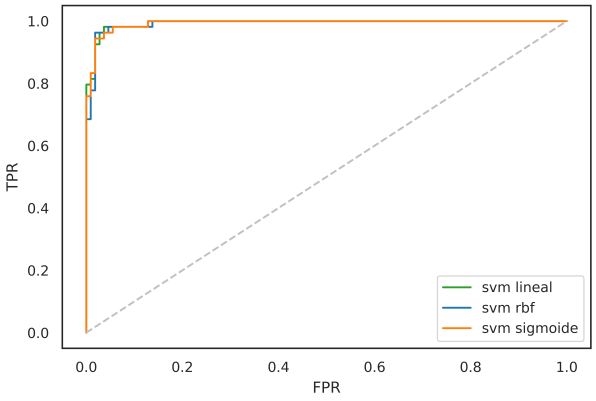

Figura 5.8: Curvas ROC de los tres modelos de máquinas de soporte vectorial con stop words y textos no lematizados

5.2.2. Modelo 2 (Naive Bayes)

Se entrenó el modelo utilizando el clasificador de naive bayes gaussiano, es decir, las distribuciones marginales utilizadas se suponen que siguen una distribución normal. Como este algoritmo no posee parámetros a optimizar se omitió el proceso de validación en este caso.

Resultados en el conjunto de test sin stop words y textos lematizados

Se realizaron las predicciones sobre las 163 noticias que consta el conjunto de test y analizando la matriz de confusión resultante, se tiene que el modelo clasifica de manera correcta gran parte de las noticias pero también cometiendo una cantidad considerable de errores. Además tiene una tendencia mayor a falsos positivos que falsos negativos.

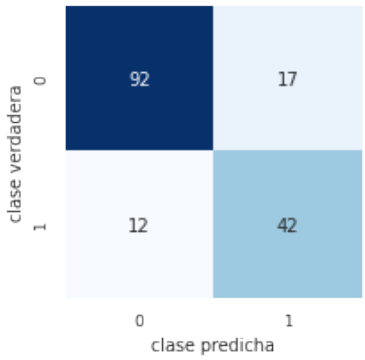

Figura 5.9: Matriz de confusión elaborada con las predicciones del conjunto de test y el clasificador de naive bayes gaussiano sin stop words y textos lematizados

Análisis ROC

El modelo entrenado presenta una buena capacidad para discriminar una noticia falsas de una real, en efecto tiene un área bajo la curva ROC de aproximadamente 81,09 %. El umbral para el que se tiene la mayor diferencia entre la tasa de falsos positivos y la tasa de verdaderos positivos es aproximadamente expresados en porcentaje 100 %. Algo interesante es que las predicciones de las probabilidades de las noticias para ese modelo solo toman los valores de 0 o 1.

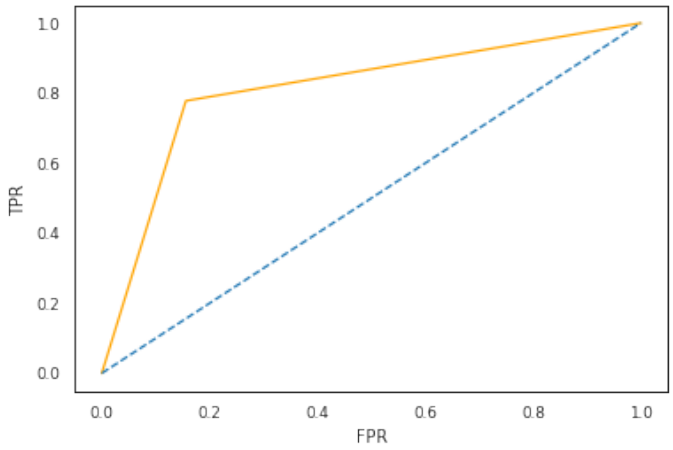

Figura 5.10: Curva ROC modelo naive bayes gaussiano sin stop words y textos lematizados

Resultados en el conjunto de test con stop words y textos no lematizados

Se realizaron las predicciones sobre las 163 noticias que consta el conjunto de test y analizando la matriz de confusión resultante, se tiene que el modelo clasifica de manera correcta gran parte de las noticias pero también cometiendo una cantidad

considerable de errores. Además tiene una tendencia mayor a falsos negativos que falsos positivos.

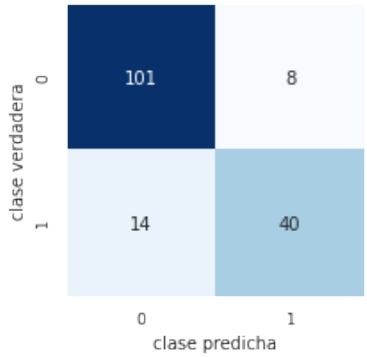

Figura 5.11: Matriz de confusión elaborada con las predicciones del conjunto de test con el clasificador de naive bayes gaussiano con stop words y textos no lematizados

Análisis ROC

El modelo entrenado presenta una buena capacidad para discriminar una noticia falsa de una real, en efecto tiene un área bajo la curva ROC de aproximadamente 83,37 %. El umbral para el que se tiene la mayor diferencia entre la tasa de falsos positivos y la tasa de verdaderos positivos es aproximadamente expresados en porcentaje 100 %. Algo interesante es que las predicciones de las probabilidades de las noticias para este modelo solo toman los valores de 0 o 1.

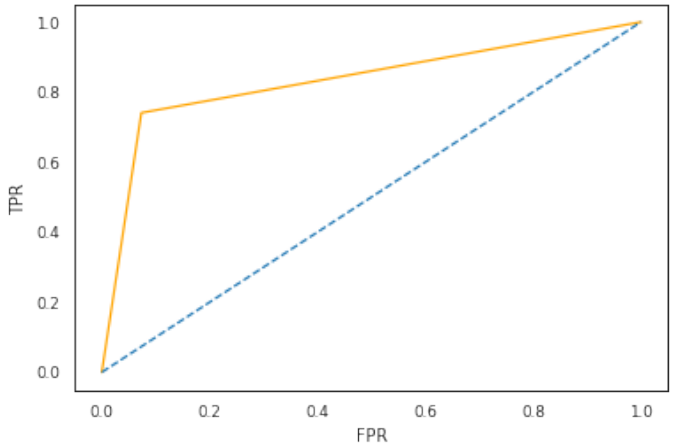

Figura 5.12: Curva ROC modelo naive bayes gaussiano con stop words y textos no lematizados

5.2.3. Modelo 3 (Bosques aleatorios)

Se entrenaron dos tipos de modelos de árboles aleatorios, con criterios de división del espacio del espacio de entrenamiento diferentes: índice de gini y entropía cruzada como se analizó en la **Sección 3.3.2**. El número de árboles de decisión entrenados sobre muestras bootstrap óptimo fue tomado a partir de una secuencia de valores de 1 a 700 cada 25.

Número de árboles óptimo sin stop words y textos lematizados

A medida que se incrementa el número de árboles de decisión el valor del error proporcionado por la función de perdida para los conjuntos de entrenamiento, test y observaciones *OOB*, decrece rápidamente para los primeros valores, específicamente de 1 a 100 árboles. A partir de los 475 árboles ya no se tiene un cambio significativo el error *OOB*, y por otro lado, el error del conjunto de validación comienza a incrementarse a partir de ese número de árboles, por lo que se tomó como parámetro óptimo 475 árboles de decisión para el modelo con índice de gini, análogamente para el modelo con entropía cruzada se tiene como parámetro óptimo 400 árboles de decisión.

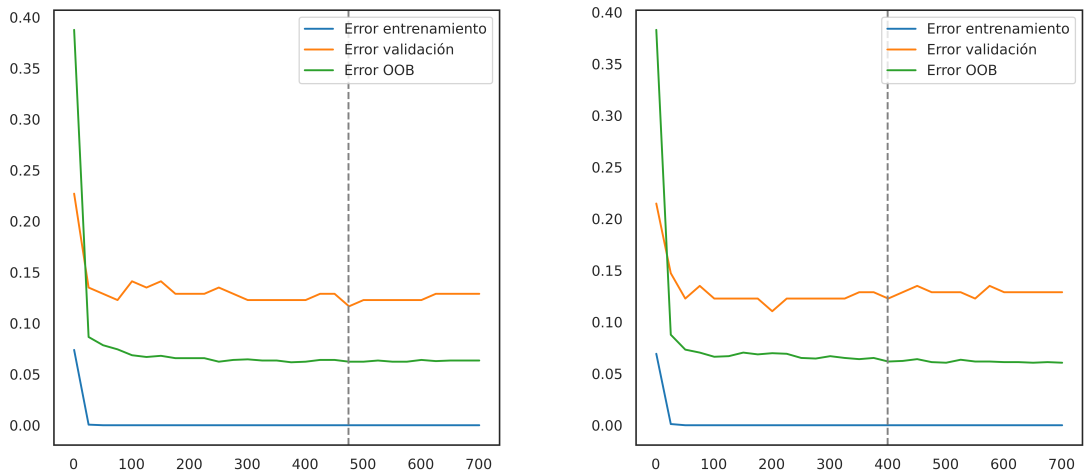

Figura 5.13: Número óptimo de árboles de decisión para bosques aleatorios sin stop words y textos lematizados. Izquierda: índice de gini. derecha: entropía cruzada

Resultados en el conjunto de test

En base a las predicciones hechas sobre las noticias del conjunto de test se obtuvo la matriz de confusión, se tiene que ambos modelos tienen un buen despeño clasificando correctamente la mayor parte de las noticias, pero por otro lado los modelos tienen tendencia a cometer falsos negativos a comparación de los falsos positivos que son muy reducidos.

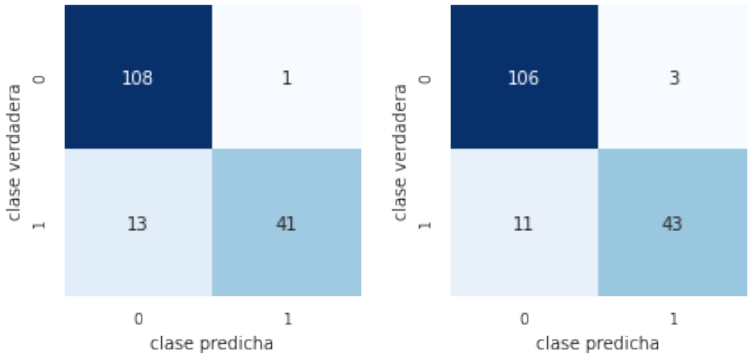

Figura 5.14: Matrices de confusión elaboradas con las predicciones del conjunto de test stop words y textos lematizados. Izquierda: índide de gini. Derecha: entropía cruzada

Análisis ROC

El modelo entrenado presenta una gran capacidad para discriminar una noticia falsas de una real ya que se tiene áreas bajo la curva ROC para los modelos con índice de gini y entropía cruzada de aproximadamente 97,47 % y 97,63 % respectivamente. Los umbrales para los que se tiene la mayor diferencia entre la tasa de falsos positivos y la tasa de verdaderos positivos es aproximadamente expresado en porcentaje 36,42 % para el modelo con índice de gini y 36,5 % para el modelo con entropía cruzada.

Número de árboles óptimo con stop words y textos no lematizados

De igual forma, a medida que se incrementa el número de árboles de decisión el valor del error proporcionado por la función de perdida para los conjuntos de entrenamiento, test y observaciones *OOB*, decrece rápidamente para los primeros valores, específicamente de 1 a 100 árboles. A partir de los 450 árboles ya no se tiene un cambio significativo el error *OOB*, y por otro lado, el error del conjunto de validación comienza a incrementarse a partir de ese número de árboles, por lo que

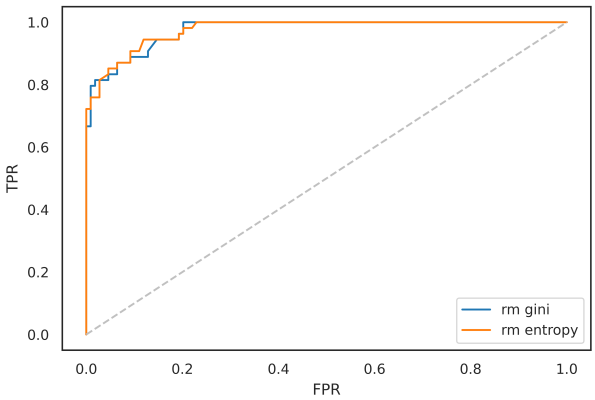

Figura 5.15: Curvas ROC de los modelos de bosques aleatorios sin stop words y textos lematizados

se tomó como parámetro óptimo 475 árboles de decisión para el modelo con índice de gini, análogamente para el modelo con entropía cruzada se tiene como parámetro óptimo 350 árboles de decisión.

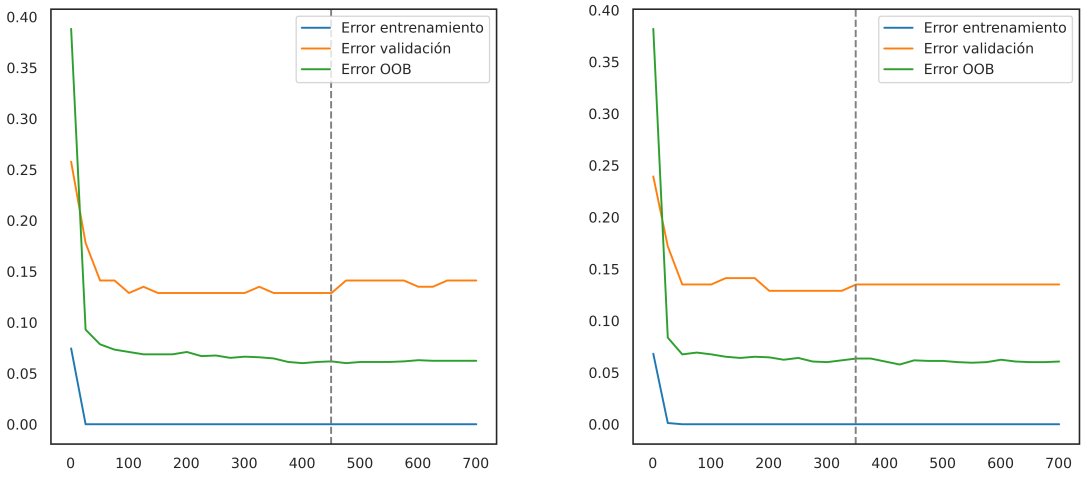

Figura 5.16: Número óptimo de árboles de decisión para bosques aleatorios con stop words y textos no lematizados. Izquierda: índice de gini. derecha: entropía cruzada

Resultados en el conjunto de test

En base a las predicciones hechas sobre las noticias del conjunto de test se obtuvo la matriz de confusión, se tiene que ambos modelos tienen un buen despeño clasificando correctamente la mayor parte de las noticias, pero por otro lado los modelos tienen tendencia a cometer falsos negativos a comparación de los falsos positivos

que son muy reducidos.

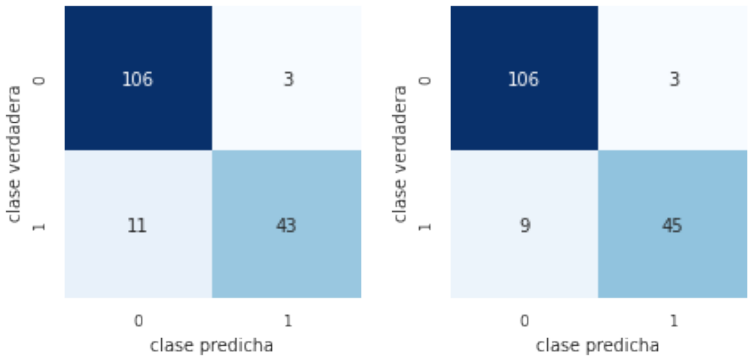

Figura 5.17: Matrices de confusión elaboradas con las predicciones del conjunto de test con stop words y textos no lematizados. Izquierda: índide de gini. Derecha: entropía cruzada

Análisis ROC

El modelo entrenado presenta una gran capacidad para discriminar una noticia falsas de una real ya que se tiene áreas bajo la curva ROC para los modelos con índice de gini y entropía cruzada de aproximadamente 98,30 % y 97,93 % respectivamente. Los umbrales para los que se tiene la mayor diferencia entre la tasa de falsos positivos y la tasa de verdaderos positivos es aproximadamente expresado en porcentaje 41,78 % para el modelo con índice de gini y 43,43 % para el modelo con entropía cruzada.

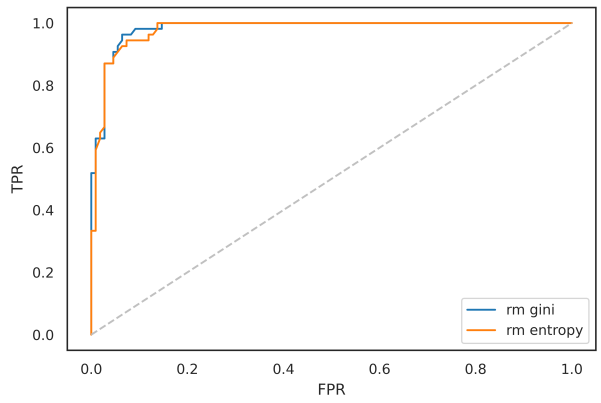

Figura 5.18: Curvas ROC de los modelos de bosques aleatorios con stop words y textos no lematizados

5.2.4. Modelo 4 (Boosting)

Se entrenó un único modelo con árboles boosting, como se analizó en la **Sección 3.2.4**, uno de los parámetros elegidos para el modelo es el uso del algoritmo AdaBoost.M1 que nos permite la librería *scikit-learn*. El número de árboles boosting entrenados sobre las modificaciones del conjunto de entrenamiento óptimo fue tomado a partir de una secuencia de valores de 1 a 1000 cada 25.

Número de árboles óptimo sin stop words y textos lematizados

El valor de la función de pérdida decrece rápidamente para modelos boosting con 1 a 200 árboles de decisión tanto para el conjunto de entrenamiento como para el conjunto de test. El error del conjunto de validación ya no cambia significativamente en su valor a partir de los 630 árboles, incluso, comienza ligeramente a incrementarse, por lo que se tomó ese número de árboles de decisión como parámetro óptimo para este modelo.

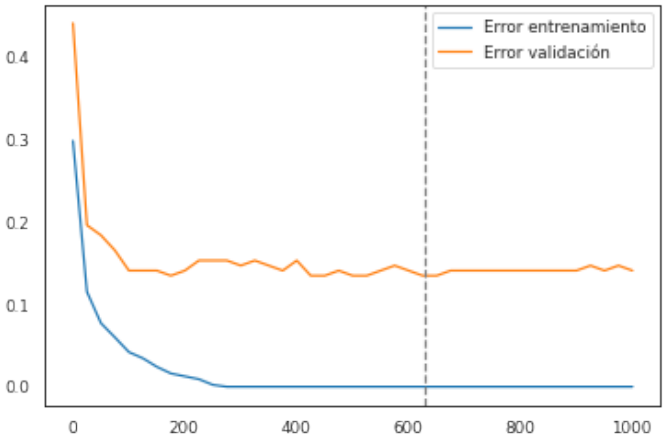

Figura 5.19: Número óptimo de árboles de decisión para modelo boosting sin stop words y textos lematizados.

Resultados en el conjunto de test

Analizando los resultados obtenidos de la matriz de confusión elaborada con las predicciones sobre las noticias del conjunto de test, tenemos que el modelo boosting tiene un desempeño deseable, ya que clasifica la mayor parte de las noticias correctamente pero con una tendencia a cometer falsos negativos, mientras que los falsos positivos son muy reducidos.

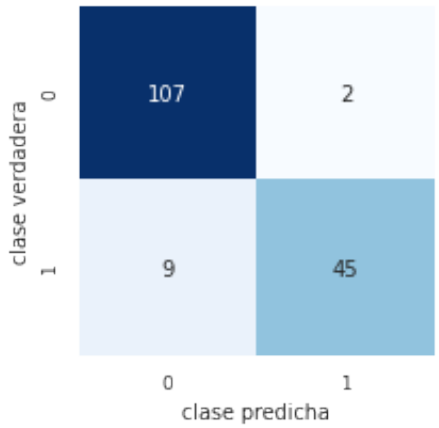

Figura 5.20: Matriz de confusión elaborada con las predicciones del conjunto de test para el modelo boosting sin stop words y textos lematizados

Análisis ROC

El modelo entrenado presenta una buena capacidad para discriminar una noticia falsas de una real, en efecto tiene un área bajo la curva ROC de aproximadamente 97,27 %. El umbral para el que se tiene la mayor diferencia entre la tasa de falsos positivos y la tasa de verdaderos positivos es aproximadamente expresado en porcentaje 23,07 %.

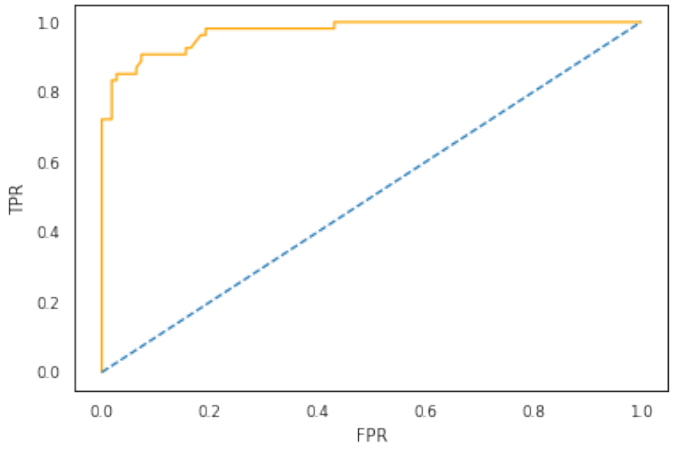

Figura 5.21: Curva ROC modelo boosting sin stop words y textos lematizados

Número de árboles óptimo con stop words y textos no lematizados

El valor de la función de pérdida tiene un decrecimiento más lento tanto para el conjunto de entrenamiento como para el conjunto de test mientras aumenta el número de árboles de 1 a 400 en comparación del modelo sin stop words y textos

lematizados. El error del conjunto de validación llega a un valor mínimo para los 900 arboles de decisión, por lo que se tomó ese número de árboles de decisión como parámetro óptimo para este modelo.

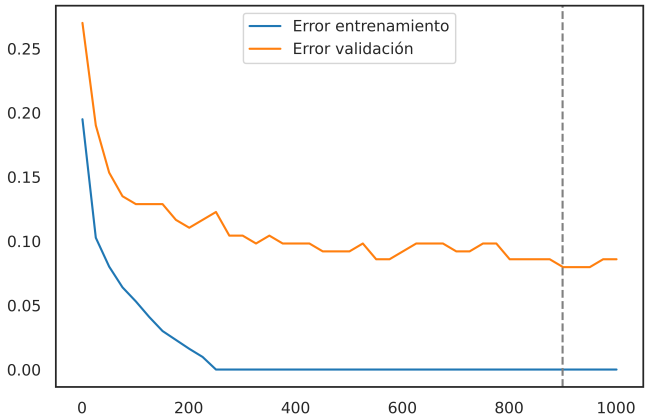

Figura 5.22: Número óptimo de árboles de decisión para modelo boosting con stop words y textos no lematizados.

Resultados en el conjunto de test

Analizando los resultados obtenidos de la matriz de confusión elaborada con las predicciones sobre las noticias del conjunto de test, tenemos que el modelo boosting tiene un desempeño deseable, ya que clasifica la mayor parte de las noticias correctamente con aproximadamente igual pero con una tendencia ligera a cometer falsos negativos.

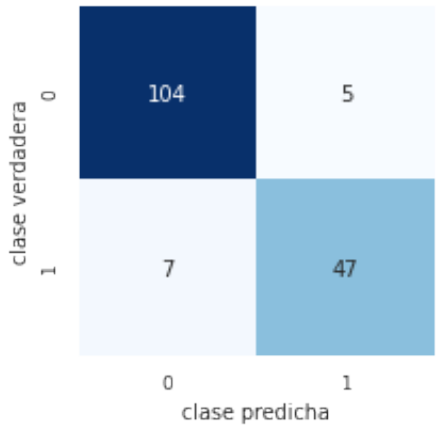

Figura 5.23: Matriz de confusión elaborada con las predicciones del conjunto de test para el modelo boosting con stop words y textos no lematizados

Análisis ROC

El modelo entrenado presenta una buena capacidad para discriminar una noticia falsas de una real, en efecto tiene un área bajo la curva ROC de aproximadamente 97,60 %. El umbral para el que se tiene la mayor diferencia entre la tasa de falsos positivos y la tasa de verdaderos positivos es aproximadamente expresado en porcentaje 12,89 %.

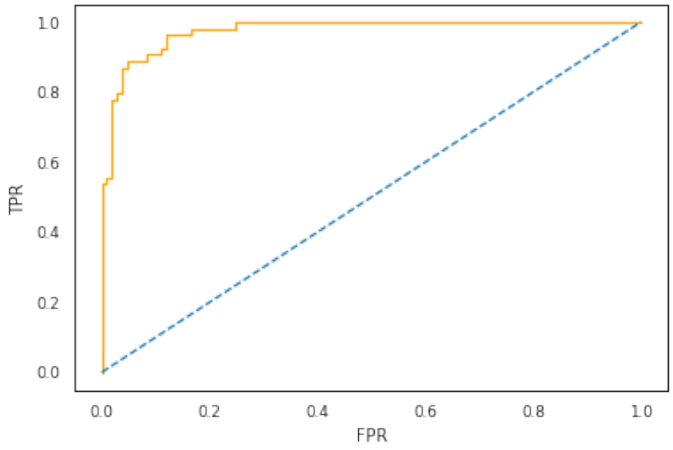

Figura 5.24: Curva ROC modelo boosting con stop words y textos no lematizados

5.3. Comparación de modelos

Con los modelos finales se comparó el desempeño de cada uno siguiendo dos enfoques distintos: la capacidad de predicción dada por las métricas de rendimiento de la **Sección 3.4.1** y el comportamiento del error de predicción esperado estimado con validación cruzada de la **Sección 3.3.4** con el objetivo de escoger el mejor modelo de detección de noticias falsas. En esta sección no se realizó el proceso de balanceo de la base de datos para poder tener una estimación real del error de predicción esperado.

5.3.1. Error de predicción esperado

La estimación del error de predicción esperado obtenido del proceso de validación cruzada fue llevado a cabo gracias a la utilización de la librería *scikit-learn* (Pedregosa et. al, 2011) en la cual se encuentra implementado dicho algoritmo y los resultados se encuentran en la **Tabla A.1**.

Como se analizó en la **Sección 3.3.4** los valores usuales para particionar los datos son $K = 5$ y $K = 10$ por lo que se tomó estos valores para la estimación. Con $K = 5$ de las 1629 noticias del corpus aleatoriamente se tomaron 5 grupos de aproximadamente 326 noticias cada uno, con lo que se entrenó los 5 modelos con un conjunto de aproximadamente 1304 y se realizó las predicciones sobre aproximadamente 326 noticias.

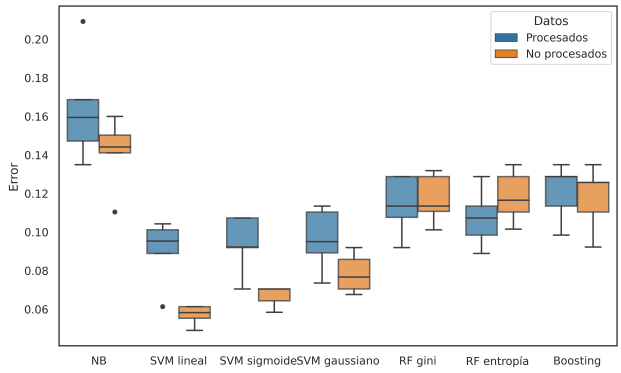

Figura 5.25: Estimación del error de predicción esperado utilizando validación cruzada con 5 grupos para cada uno de los modelos propuestos con sus variantes para textos no procesados

De forma análoga se realizó las estimaciones con $K = 10$, así, de las 1629 noticias del corpus aleatoriamente se tomaron 10 grupos de aproximadamente 163 noticias cada uno, con lo que se entrenó los 10 modelos con un conjunto de aproximadamente 1630 y se realizó las predicciones sobre aproximadamente 163 noticias.

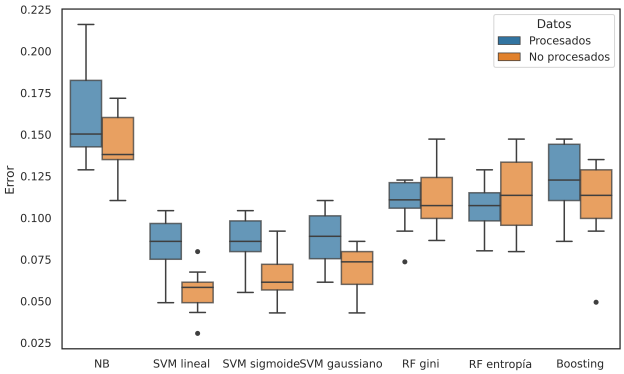

Figura 5.26: Estimación del error de predicción esperado utilizando validación cruzada con 10 grupos para cada uno de los modelos propuestos con sus variantes para textos no procesados

5.3.2. Capacidad de predicción

Para medir de manera global el rendimiento de las predicciones de los modelos se utilizó la medida *F-score* ya que como vimos en la **Sección 3.4.1** esta medida utiliza tanto la cobertura y la precisión de cada una de las clases de la variable respuesta para crear una medida más general. Todos los modelos presentan un gran poder predictivo, ya que tienen valores *F-score* que sobrepasan el 80 %, pero hay que destacar que los modelos con maquinas de soporte vectorial son los que tienen las puntuaciones más altas, en cambio, los modelos con clasificador de naive bayes gaussiano los que menos. La reducción del número de variables del conjunto de datos al momento de procesar los textos de las noticias, podría reducir el poder predicativo de los modelos, lo cual es verdad, pero es claro que para este caso la reducción es mínima, ya que los valores de *F-score* para modelos sin stop words y textos lematizados son practicante iguales a los modelos con stop words y textos no lematizados.

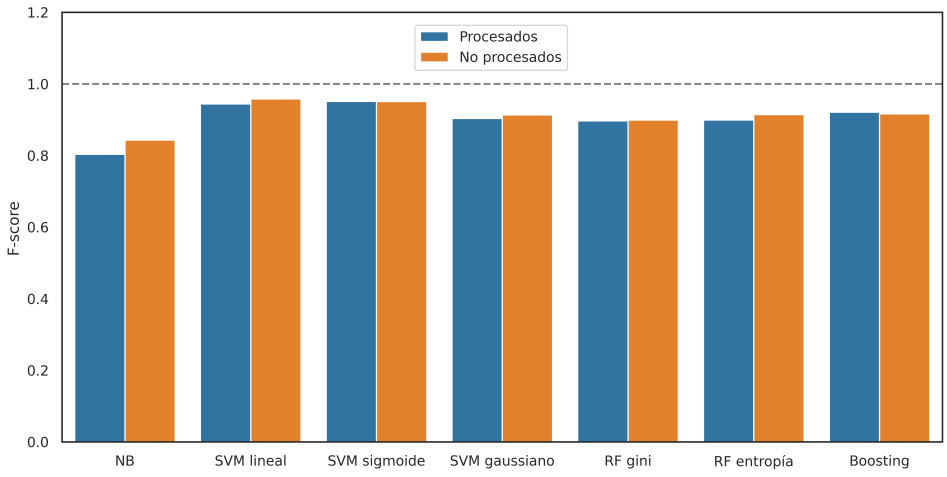

Figura 5.27: Valor de la medida F-score para cada uno de los diferentes algoritmos de clasificación con sus respectivas variaciones para datos con stop words y textos no lematizados

5.4. Análisis de resultados

En base a los resultados obtenidos a lo largo de esta sección, en general los modelos logran predecir acertadamente si una noticia es real o falsa, como se puede apreciar en la **Figura 5.27** logrando valores altos para la medida *F-score general*, sien-

do los modelos con máquinas de soporte vectorial con kernel lineal y sigmoide los que ofrecen las mejores predicciones llegando a exactitud por arriba del 95 % . Aparentemente el pre procesamiento de los textos no tienen mucha relevancia sobre el poder predictivo de los modelos, pero si analizamos la **Figura 5.26** este proceso si tiene un impacto directo sobre las estimaciones de los errores de predicción, siendo estos mucho menos variables y en valor menores para los modelos con stop words y textos no lematizados que para los modelos sin stop words y textos lematizados, lo cual está relacionado a la reducción de información consecuencia de disminuir el número de variables del conjunto de datos; esto se cumple para los modelos con máquinas de soporte vectorial y los dos modelos con clasificador de naive bayes, ya que para los modelos entrenados con textos procesados que utilizan árboles de decisión (boosting y bosques aleatorios), presentan estimaciones de los errores de predicción incluso mejores que los modelos con árboles de decisión con textos no procesados.

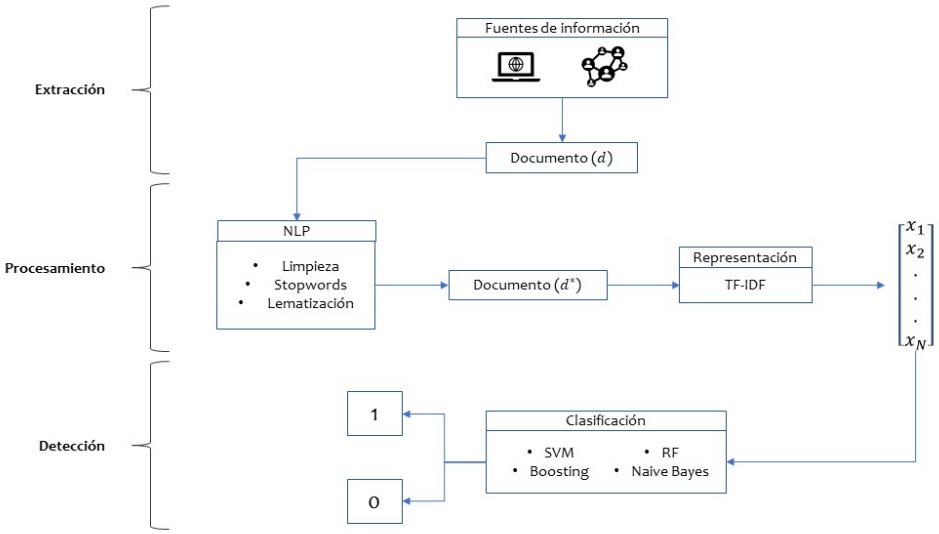

Figura 5.28: Esquema gráfico del proceso que consta el sistema de detección automática de noticias falsas en español con cada uno de sus componentes

Una vez contrastado los resultados obtenidos de cada uno de los modelos y sabiendo cuales de estos son los que mejor detectan noticias falsas, se obtuvo un sistema de identificación automática que comprende tres módulos. El primer módulo es el de *extracción*, que es obtener las noticias de forma manual o utilizando técnicas de webscrapping de las diferentes fuentes de información, estas fuentes pueden

ser las de la **Sección 4.2.1** o en general otras ya que con los modelos entrenados se puede analizar noticias de otras páginas web o diferentes redes sociales. El segundo módulo es el de *procesamiento*, en este consta el tratamiento del documento, desde su limpieza hasta su representación numérica mediante la técnica $TF\text{-}IDF$, en este caso un documento es representado en un vector de 6336 componentes ya que es el número de palabras existentes en el corpus creado. Y por último el tercer módulo es el de *detección* en el que se utilizan los distintos modelos de clasificación entrenados para identificar si una noticia es real o falsa con una precisión de al menos 90 %.

Capítulo 6

Conclusiones y Recomendaciones

El estudio realizado muestra la capacidad de los algoritmos de clasificación supervisada para lograr identificar con mas del 90 % de precisión noticias falsas que circulan diariamente en internet, especialmente en redes sociales por parte de páginas de sátira política enfocadas en problemáticas ecuatorianas, que si bien su objetivo no es malicioso sino humorístico, pero la información que se crea y es compartida puede ser mal interpretada fuera de contexto desinformando a las personas. Las redes sociales han demostrado ser una potente herramienta de comunicación masiva, pero también son una excelente fuente para obtener datos tanto de los usuarios como de las páginas creadas en estas plataformas, la información extraída de Facebook fue de vital importancia para crear el corpus de noticias utilizado para entrenar los modelos, ya que al no contar con una base de datos de noticias falsas verificada por expertos, estas páginas fueron la única fuente fácilmente verificable para obtener noticias con contenido falso y accesible para cualquier persona.

La representación numérica de textos en forma de vectores fue un aspecto muy importante en este trabajo, ya que fue la pieza principal en la que se fundamentó para obtener los datos necesarios para el proceso de entrenamiento, la técnica frecuencia de término inversa logró captar la estructura que caracteriza a una noticia falsa ya que como vimos esta se basa en calcular la importancia que tiene cada una de las palabras tanto a nivel de la noticia como a un nivel más general dentro del corpus. El estilo de escritura de las noticias falsas es similar al de una noticia real ya que pretende simular ser una pero el tipo de palabras que se utilizan para elaborarlas son diferentes en especial por que hacen uso de palabras propias de la jerga ecuatoriana y es precisamente esto lo que la frecuencia de término inversa logra representar numéricamente. El número de variables resultantes de la representación

numérica de las noticias del corpus para este caso, estuvo cercana a las diez mil lo que se traduce a un gran gasto computacional, por lo que procesar previamente los textos fue de gran ayuda para reducir los tiempos de ejecución de los algoritmos sin perder casi nada información y manteniendo un excelente rendimiento de los modelos al realizar predicciones.

Luego de analizar los resultados obtenidos del proceso de entrenamiento y de las predicciones sobre el conjunto de test, es claro que las máquinas de soporte vectorial logran sobreponerse a los demás algoritmos de clasificación supervisada, el mejor de estos es el que usa kernel lineal con un 96,32 % de precisión, por otro lado, el clasificador de naive bayes es el que menor rendimiento ofrece y esto se debe a que este algoritmo realiza una suposición fuerte que es que los datos se comportan con una distribución normal pero generalmente esto no siempre se cumple, sin embargo ofrece buenos resultados con un 86,5 % de precisión el mejor de ellos. La mayor diferencia entre los modelos radica en la estimación del error de predicción esperado, como se pudo analizar el comportamiento de los errores estimados de las máquinas de soporte vectorial es bastante superior al resto de modelos, con promedios del 5 % al 10 % y una variabilidad de 0,5 % a 1,5 %, por otro lado el clasificador de naive bayes presenta los errores más grandes con promedios del 15 % al 17 % y una variabilidad de 1,5 % a 3 %.

Lo que se ha logrado es crear un sistema de identificación automática de noticias falsas para noticias provenientes de páginas web y redes sociales, con tres módulos que pueden ser utilizados independientemente para extraer noticias de las páginas oficiales de El Universo y El Comercio con técnicas de webscrapping, procesar cualquier tipo de textos con procesamiento del lenguaje natural y clasificar textos relacionados a noticias como reales o falsos de manera precisa con modelos de aprendizaje estadístico. Como el estudio realizado se concentró para noticias dentro del ámbito ecuatoriano este tiene relevancia dentro del mismo ya que la verificación de contenido político y contenido en general lo realiza la organización *Ecuador Chequea*, el inconveniente del fact checking tradicional es que la verificación conlleva un periodo de ejecución, recolección de información e involucra un equipo especializado de trabajo para analizar la veracidad de un grupo de noticias, la mayor ventaja de crear un modelo automático de detección de noticias falsas es agilizar este proceso, que sea escalable a la cantidad de información que circula en linea y esté accesible a cualquier persona, si bien el fact checking automático no supera en calidad al trabajo realizado por los fact ckeckers, es una herramienta alternativa de mucha ayuda para identificar noticias falsas mucho más rápido, que puede ser usado tanto por la

ciudadanía en general como también por organizaciones que realizan fact checking y que sea una herramienta extra de la que pueden valerse para verificar el contenido de una notica.

Este estudio se lo ha realizado con datos públicos de redes sociales y de páginas de periódicos nacionales, se podría profundizar en la investigación creando un modelo más sofisticado con la ayuda de expertos en fact-checking para crear un corpus de noticias mas grande abordando más temáticas además de la sátira política con el fin de generalizar la detección de noticias falsas. El modelo creado tiene una gran aplicación dentro del campo tecnológico, siendo así que puede ser utilizado como una herramienta de ayuda tanto para personas como para empresas a combatir de forma directa la desinformación, especialmente en situaciones de crisis en la que las redes sociales juegan un papel fundamental dentro de la comunicación, con lo que se lograría un mejor entendimiento del comportamiento que tienen los usuarios en redes sociales durante eventos críticos. Este trabajo es un avance a lo que respecta la detección de noticias falsas y el procesamiento del lenguaje natural en español, por lo que se recomienda profundizar con diferentes metodologías como la utilización de redes neuronales artificiales y añadiendo elementos como imágenes, videos o audios y nuevas técnicas de procesamiento de texto para el idioma español.

Bibliografía

[1] Allcott, H. y Gentzkow, M. Social media and fake news in the 2016 election. *Journal of Economic Perspectives*, 31:211–236, 05 2017.

[2] Baeza-Yates, R. Challenges in the interaction of information retrieval and natural language processing. En Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, págs. 445–456, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[3] Bird, S., Klein, E., y Loper, E. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edición, 2009.

[4] Bondielli, A. y Marcelloni, F. A survey on fake news and rumour detection techniques. *Information Sciences*, 497, 05 2019.

[5] Bowyer, K. W., Chawla, N. V., Hall, L. O., y Kegelmeyer, W. P. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.

[6] Ciampaglia, G. L., Shiralkar, P., Rocha, L. M., Bollen, J., Menczer, F., y Flammini, A. Computational fact checking from knowledge networks. *CoRR*, abs/1501.03471, 2015.

[7] Clément, L., Gerdes, K., y Marlet, R. A grammar correction algorithm. En de Groote, P., Egg, M., y Kallmeyer, L., editors, *Formal Grammar*, págs. 47–63, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[8] Deng, L. y Wiebe, J. How can NLP tasks mutually benefit sentiment analysis? a holistic approach to sentiment analysis. En *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, págs. 53–59, San Diego, California, June 2016. Association for Computational Linguistics.

[9] Hastie, T., Tibshirani, R., y Friedman, J. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edición, 2009.

[10] Honnibal, M. y Montani, I. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[11] Howard, J. y Ruder, S. Universal language model fine-tuning for text classification, 2018.

[12] James, G., Witten, D., Hastie, T., y Tibshirani, R. *An Introduction to Statistical Learning – with Applications in R*, volume 103 of *Springer Texts in Statistics*. Springer, New York, 2013.

[13] Kapoor, N., Vishal, S., y K. S., K. Movie recommendation system using nlp tools. En *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, págs. 883–888, 2020.

[14] Lazer, D., Baum, M., Benkler, Y., Berinsky, A., Greenhill, K., Menczer, F., Metzger, M., Nyhan, B., Pennycook, G., Rothschild, D., Schudson, M., Sloman, S., Sunstein, C., Thorson, E., Watts, D., y Zittrain, J. The science of fake news. *Science*, 359:1094–1096, 03 2018.

[15] Lemaître, G. *Illustration of the sample generation in the over-sampling algorithm*, imbalanced-learn, 2017. `https://imbalanced-learn.readthedocs.io/en/stable/auto_examples/over-sampling/plot_illustration_generation_sample.html`.

[16] Lemaître, G., Nogueira, F., y Aridas, C. K. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[17] López-Borrull, A., Vives-Gràcia, J., y Badell, J.-I. Fake news, ¿amenaza u oportunidad para los profesionales de la información y la documentación? *El Profesional de la Información*, 27:1346, 12 2018.

[18] Mohri, M., Rostamizadeh, A., y Talwalkar, A. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2 edición, 2018.

[19] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., y Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] Posadas Durán, J., Gomez Adorno, H., Sidorov, G., y Moreno, J. Detection of fake news in a new corpus for the spanish language. *Journal of Intelligent & Fuzzy Systems*, 36:4869–4876, 05 2019.

[21] Pulido, C. M., Ruiz-Eugenio, L., Redondo-Sama, G., y Villarejo-Carballido, B. A new application of social impact in social media for overcoming fake news in health. *International journal of environmental research and public health*, 17(7):2430, Apr 2020.

[22] Rodríguez-Fernández, L. Desinformación y comunicación organizacional: estudio sobre el impacto de las fake news. *Revista Latina de Comunicacion Social*, 74:1714–1728, 12 2019.

[23] Rosenblatt, F. *Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms*. Spartan Books,Washington D.C., 1962.

[24] Shelomi, M. *Opinion: Using Pokémon to Detect Scientific Misinformation*, The Scientist Magazine, 2020. https://www.the-scientist.com/critic-at-large/opinion-using-pokmon-to-detect-scientific-misinformation-68098.

[25] Singhania, S., Fernandez, N., y Rao, S. 3han: A deep neural network for fake news detection. 11 2017.

[26] Soll, J. *The Long and Brutal History of Fake News: Bogus news has been around a lot longer than real news. And it's left a lot of destruction behind*, Politico Magazine, 2016. `https://www.politico.com/magazine/story/2016/12/fake-news-history-long-violent-214535`.

[27] Vajjala, S., Majumder, B., Gupta, A., y Surana, H. *Practical Natural Language Processing*. O'Reilly Media, Inc., 1st edición, 2020.

[28] Zaki, M. J. y Jr, W. M. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, USA, 2014.

[29] Zhang, X. y Ghorbani, A. An overview of online fake news: Characterization, detection, and discussion. *Information Processing & Management*, 57, 03 2019.

[30] Zhou, X. y Zafarani, R. A survey of fake news: Fundamental theories, detection methods, and opportunities. *ACM Comput. Surv.*, 53(5), September 2020.

Apéndice A

Resultados de la evaluación de los modelos

	$K = 5$		$K = 10$	
Modelo	promedio	desviación	promedio	desviación
NB	0.8361	0.0253	0.8361	0.0278
SVM lineal	0.9098	0.0154	0.9171	0.0165
SVM sigmoide	0.9061	0.0135	0.9147	0.0146
SVM gaussiano	0.9036	0.0146	0.9122	0.015
RF gini	0.8901	0.0144	0.8901	0.0151
RF entropía	0.8914	0.016	0.895	0.016
Boosting	0.8797	0.0138	0.8815	0.0214
NB *	0.8588	0.0167	0.8551	0.0179
SVM lineal *	0.9429	0.0046	0.9441	0.013
SVM sigmoide *	0.9331	0.0048	0.9362	0.0145
SVM gaussiano *	0.9214	0.0092	0.9306	0.014
RF gini *	0.8828	0.0115	0.8877	0.0179
RF entropía *	0.8815	0.0121	0.8864	0.0221
Boosting *	0.8822	0.015	0.8908	0.0243

Tabla A.1: Precisión esperada de los modelos a partir los errores de predicción esperados utilizando validación cruzada

Modelo	Err	Acc	$prec_F$	$prec_R$	TPR	TFR	FNR	FPR	F_F	F_R
SVM lineal	4.91 %	95.09 %	94.23 %	95.5 %	90.74 %	97.25 %	9.26 %	2.75 %	92.45 %	96.36 %
SVM sigmoide	4.29 %	95.71 %	94.34 %	96.36 %	92.59 %	97.25 %	7.41 %	2.75 %	93.46 %	96.80 %
SVM rbf	7.98 %	92.02 %	100 %	89.34 %	75.93 %	100 %	24.07 %	0 %	86.32 %	94.37 %
SVM lineal *	3.68 %	96.32 %	96.15 %	96.4 %	92.59 %	98.17 %	7.41 %	1.83 %	94.34 %	97.27 %
SVM sigmoide *	4.29 %	95.71 %	96.08 %	95.54 %	90.74 %	98.17 %	9.26 %	1.83 %	93.33 %	96.83 %
SVM rbf *	7.36 %	92.64 %	95.65 %	91.45 %	81.48 %	98.17 %	18.52 %	1.83 %	88 %	94.69 %
NB gaussiano	17.79 %	82.21 %	71.19 %	88.46 %	77.78 %	84.4 %	22.22 %	15.6 %	74.34 %	86.38 %
NB gaussiano *	13.5 %	86.5 %	83.33 %	87.83 %	74.07 %	92.66 %	25.93 %	7.34 %	78.43 %	90.19 %
RF gini	8.59 %	91.41 %	97.62 %	89.26 %	75.93 %	99.08 %	24.07 %	0.92 %	85.42 %	93.91 %
RF entropy	8.59 %	91.41 %	93.48 %	90.6 %	79.63 %	97.25 %	20.37 %	2.75 %	86 %	93.81 %
RF gini *	8.59 %	91.41 %	93.48 %	90.6 %	79.63 %	97.25 %	20.37 %	2.75 %	86 %	93.81 %
RF entropy *	7.36 %	92.64 %	93.75 %	92.17 %	83.33 %	97.25 %	16.67 %	2.75 %	88.24 %	94.64 %
Boosting	6.75 %	93.25 %	95.74 %	92.24 %	83.33 %	98.17 %	16.67 %	1.83 %	89.11 %	95.11 %
Boosting *	7.36 %	92.64 %	90.38 %	93.69 %	87.04 %	95.41 %	12.96 %	4.59 %	88.68 %	94.54 %

Tabla A.2: Resultados de las métricas de evaluación para todos los modelos entrenados

Apéndice B

Código

B.0.1. Textos sin stop words y textos lematizados

```
import pandas as pd
import numpy as np
import re
import string
import functools
import matplotlib.pyplot as plt
import nltk
import textblob
import spacy
import seaborn as sns

from collections import Counter
from nltk.tokenize import word_tokenize
from sw import palabras_parada
from sklearn import metrics
from nltk.corpus import stopwords
from wordcloud import WordCloud
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report,
    confusion_matrix, plot_confusion_matrix
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from imblearn.over_sampling import SMOTE
from sklearn.feature_extraction.text import CountVectorizer,
   TfidfVectorizer
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

nltk.download("stopwords")
nlp_md = spacy.load("es_core_news_md")
sw=stopwords.words('spanish')+palabras_parada
sns.set_style("white")
sns.set_context("paper")

"""# Funciones"""

def remover_links(x):
    l=re.compile('https:\S*')
    return l.sub('',x.lower())
def remover_puntc(x):
    p=re.compile('[^\w\s]')
    return p.sub(' ',x.lower())
def remover_caract_ext(x):
    c=re.compile('<\S*>')
    return c.sub('',x.lower())
def remover_carac_espacio(x):
    e=re.compile('(\r)|(\n)')
    return e.sub('',x.lower())
def remover_num(x):
    n=re.compile('[0-9]*')
    return n.sub('',x.lower())
def remover_etiq(x):
    e=re.compile('@\S*')
    return e.sub('',x.lower())
```

```
def remover_hash(x):
    h=re.compile('#\S*')
    return h.sub('',x.lower())
def remover_single(x):
    h=re.compile('\s\w\s')
    return h.sub(' ',x.lower())
def remover_spaces(x):
    return " ".join(x.split())

def remover(x):
    aux=x
    aux = remover_etiq(aux)
    aux = remover_links(aux)
    aux = remover_hash(aux)
    aux = remover_caract_ext(aux)
    aux = remover_carac_espacio(aux)
    aux = remover_puntc(aux)
    aux = remover_num(aux)
    aux = remover_single(aux)
    aux = remover_spaces(aux)
    return aux.strip()

def lematizar_md(x):
    return ' '.join([tok.lemma_.lower() for tok in nlp_md(x)
       ])

def no_sw(x):
    return ' '.join([t for t in x.split() if t not in sw])

def plot_roc_curve(fpr, tpr,auc):
    plt.plot(fpr, tpr, color='orange')
    plt.plot([0, 1], [0, 1], color=(0.12156862745098039,
       0.4666666666666667, 0.7058823529411765), linestyle
       ='--')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
```

```
    plt.show()

def plot_roc_curves(fpr1,tpr1,fpr2,tpr2,fpr3,tpr3):
    fig = plt.figure()
    palette = plt.get_cmap('tab10')
    plt.plot(fpr1, tpr1,color=palette(2),label='svm lineal')
    plt.plot(fpr2, tpr2,color=palette(0),label='svm rbf')
    plt.plot(fpr3, tpr3,color=palette(1), label='svm sigmoide
       ')
    plt.plot([0, 1], [0, 1], color='silver', linestyle='--')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.legend()
    fig.savefig('svm_roc', dpi=800)
    plt.show()

def plot_roc_curves_rm(fpr1,tpr1,fpr2,tpr2):
    fig = plt.figure()
    palette = plt.get_cmap('tab10')
    plt.plot(fpr1, tpr1,color=palette(0),label='rm gini')
    plt.plot(fpr2, tpr2,color=palette(1),label='rm entropy')
    plt.plot([0, 1], [0, 1], color='silver', linestyle='--')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.legend()
    fig.savefig('rm_roc', dpi=800)
    plt.show()

def count_words(texto):
    return len(texto.split())

"""#Carga de Datos"""

data = pd.read_csv('base_noticias.csv')
data
```

```
"""# EDA textos"""

plt.figure(figsize=(4,5))
v=len(data['Tipo'][data['Tipo']==0])
f=len(data['Tipo'][data['Tipo']==1])
p=sns.countplot(x = 'Tipo', data = data)
p.set(xlabel='Tipo de noticia', ylabel='Frecuencia')
p.figure.savefig("data_dist.png",dpi=800)
'Verdaderas: {:.2f}% Falsas: {:.2f}%'.format(v/(v+f), f/(v+f)
    )

p=sns.countplot(y = 'Pagina', data = data,order =data['Pagina
    '].value_counts().index)
p.set(xlabel='Total de noticias', ylabel='')
p.figure.savefig("fuentes.png")
plt.show()

texto = data['Noticia']
texto

textos_len = list(map(count_words,texto))
base_len = pd.DataFrame({'Longitud':textos_len,'Tipo':data['
    Tipo']})

plt.figure(figsize=(4,6))
p=sns.boxplot(x='Tipo', y='Longitud', data=base_len,linewidth
    =0.75)
p.set(ylabel='N mero de palabras')
p.figure.savefig("text_boxplot.png",dpi=800)
plt.show()

plt.figure(figsize=(10,5))
p=sns.displot(x="Longitud", hue="Tipo",data=base_len, legend=
    False)
p.set(xlabel='N mero de palabras',ylabel='')
p.despine(left=True)
```

```
p.savefig("dist_text.png",dpi=800)
plt.show()

base_len.groupby('Tipo').agg({'Longitud': ['mean','std']})

"""# Procesamiento de texto

## Limpiar puntuaci n, hashtags, etiquetas, links.
"""

texto = list(map(remover,texto))

"""## Lematizar"""

texto = list(map(lematizar_md,texto))

texto = list(map(remover,texto))

"""## Stopwords"""

texto_sin_sw = list(map(no_sw,texto))

"""# Representaci n TD-IDF"""

vector = TfidfVectorizer()
vector.fit(texto_sin_sw)
data_pv_dm=vector.transform(texto_sin_sw).toarray()
pd.DataFrame(data_pv_dm)

"""# Entrenamiento

## Conjuntos de datos
"""

X=data_pv_dm
Y=np.array(data['Tipo'])
```

```
X_train_sm, X_test, Y_train_sm, Y_test = train_test_split(X,
    Y, test_size = 0.2,random_state=123)
X_val, X_test, Y_val, Y_test= train_test_split(X_test, Y_test
    , test_size=0.5, random_state=111)
smt = SMOTE(random_state=0,k_neighbors=5)
X_train, Y_train = smt.fit_sample(X_train_sm, Y_train_sm)

X_aum = pd.concat([pd.DataFrame(X_train),pd.DataFrame(X_val),
    pd.DataFrame(X_test)], axis=0, ignore_index=True)
Y_aum = pd.concat([pd.DataFrame(Y_train),pd.DataFrame(Y_val),
    pd.DataFrame(Y_test)], axis=0, ignore_index=True)
data_aum = pd.concat([X_aum,Y_aum], axis=1, ignore_index=True
    )
data_aum = data_aum.rename(columns={6336:'Tipo'})

plt.figure(figsize=(4,5))
v=len(data_aum['Tipo'][data_aum['Tipo']==0])
f=len(data_aum['Tipo'][data_aum['Tipo']==1])
p=sns.countplot(x = 'Tipo', data = data_aum)
p.set(xlabel='Tipo de noticia', ylabel='Frecuencia')
p.figure.savefig("data_dist_aum.png",dpi=800)
'Verdaderas: {:.2f}% Falsas: {:.2f}%'.format(v/(v+f), f/(v+f)
    )

"""## SVM

### Flexibilidad
"""

error = []
C = [0.01,0.5,1,5,25,50,100]
for i in C:
    model_SVM = SVC(kernel='sigmoid',tol=0.0001,C=i,
        random_state=0)
    model_SVM.fit(X_train, Y_train)
    Y_pred = model_SVM.predict(X_val)
```

```
    error.append(metrics.zero_one_loss(Y_val, Y_pred))

results_sigmoid = pd.DataFrame({'C
   ':['0.01','0.5','1','5','25','50','100'],'Error':error})
plt.figure(figsize=(3,5))
p=sns.barplot(x='C',y = 'Error', data = results_sigmoid,order
    =results_sigmoid.sort_values('Error',ascending=False).C)
p.set(ylabel='Loss')
plt.savefig("svm_sigmoid_C.png",dpi=800)
plt.show()

error = []
C = [0.01,0.5,1,5,25,50,100]
for i in C:
    model_SVM = SVC(kernel='rbf',tol=0.0001,C=i,random_state
       =0)
    model_SVM.fit(X_train, Y_train)
    Y_pred = model_SVM.predict(X_val)
    error.append(metrics.zero_one_loss(Y_val, Y_pred))

plt.figure(figsize=(3,5))
results_rbf = pd.DataFrame({'C
   ':['0.01','0.5','1','5','25','50','100'],'Error':error})
p=sns.barplot(x='C',y = 'Error', data = results_rbf,order =
   results_rbf.sort_values('Error',ascending=False).C)
p.set(ylabel='Loss')
plt.savefig("svm_rbf_C.png",dpi=800)
plt.show()

error = []
C = [0.01,0.5,1,5,25,50,100]
for i in C:
    model_SVM = SVC(kernel='linear',tol=0.0001,C=i,
       random_state=0)
    model_SVM.fit(X_train, Y_train)
    Y_pred = model_SVM.predict(X_val)
```

```
    error.append(metrics.zero_one_loss(Y_val, Y_pred))

plt.figure(figsize=(3,5))
results_linear = pd.DataFrame({'C
   ':['0.01','0.5','1','5','25','50','100'],'Error':error})
p=sns.barplot(x='C',y = 'Error', data = results_linear,order
   =results_linear.sort_values('Error',ascending=False).C)
p.set(ylabel='Loss')
plt.savefig("svm_linear_C.png",dpi=800)
plt.show()

"""### Final"""

model_SVM_sigmoid = SVC(kernel='sigmoid',tol=0.00001,
   random_state=0,probability=True,C=0.5)
model_SVM_sigmoid.fit(X_train, Y_train)
Y_pred_SVM_sigmoid = model_SVM_sigmoid.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_SVM_sigmoid))
print('Accuracy:',metrics.accuracy_score(Y_test,
   Y_pred_SVM_sigmoid))
print('Recall:',metrics.recall_score(Y_test,
   Y_pred_SVM_sigmoid))
print('Precision:',metrics.precision_score(Y_test,
   Y_pred_SVM_sigmoid))

model_SVM_rbf = SVC(kernel='rbf',tol=0.00001,random_state=0,
   probability=True,C=1)
model_SVM_rbf.fit(X_train, Y_train)
Y_pred_SVM_rbf = model_SVM_rbf.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_SVM_rbf))
print('Accuracy:',metrics.accuracy_score(Y_test,
   Y_pred_SVM_rbf))
print('Recall:',metrics.recall_score(Y_test, Y_pred_SVM_rbf))
print('Precision:',metrics.precision_score(Y_test,
   Y_pred_SVM_rbf))
```

```
model_SVM_linear = SVC(kernel='linear',tol=0.00001,
    random_state=0,probability=True,C=0.5)
model_SVM_linear.fit(X_train, Y_train)
Y_pred_SVM_linear = model_SVM_linear.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_SVM_linear))
print('Accuracy:',metrics.accuracy_score(Y_test,
    Y_pred_SVM_linear))
print('Recall:',metrics.recall_score(Y_test,
    Y_pred_SVM_linear))
print('Precision:',metrics.precision_score(Y_test,
    Y_pred_SVM_linear))

probs_sigmoid = model_SVM_sigmoid.predict_proba(X_test)
probs_sigmoid = probs_sigmoid[:,1]
auc_sigmoid = roc_auc_score(Y_test, probs_sigmoid)
fpr_sigmoid, tpr_sigmoid, thresholds_sigmoid = roc_curve(
    Y_test, probs_sigmoid)
plot_roc_curve(fpr_sigmoid, tpr_sigmoid,auc_sigmoid)
oid = np.argmax(np.abs(tpr_sigmoid - fpr_sigmoid))
print('ROC:',auc_sigmoid,'threshold:',thresholds_sigmoid[oid
    ],'TPR:',tpr_sigmoid[oid],'FPR:',fpr_sigmoid[oid])

probs_rbf = model_SVM_rbf.predict_proba(X_test)
probs_rbf = probs_rbf[:,1]
auc_rbf = roc_auc_score(Y_test, probs_rbf)
fpr_rbf, tpr_rbf, thresholds_rbf = roc_curve(Y_test,
    probs_rbf)
plot_roc_curve(fpr_rbf, tpr_rbf,auc_rbf)
oid = np.argmax(np.abs(tpr_rbf - fpr_rbf))
print('ROC:',auc_rbf,'threshold:',thresholds_rbf[oid],'TPR:',
    tpr_rbf[oid],'FPR:',fpr_rbf[oid])

probs_linear = model_SVM_linear.predict_proba(X_test)
probs_linear = probs_linear[:,1]
auc_linear = roc_auc_score(Y_test, probs_linear)
```

```
fpr_linear, tpr_linear, thresholds_linear = roc_curve(Y_test,
    probs_linear)
plot_roc_curve(fpr_linear, tpr_linear,auc_linear)
oid = np.argmax(np.abs(tpr_linear - fpr_linear))
print('ROC:',auc_linear,'threshold:',thresholds_linear[oid],'
   TPR:',tpr_linear[oid],'FPR:',fpr_linear[oid])

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_SVM_linear),
   annot=True,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_svm_linear.png",dpi=800)
plt.show()
cf_svm_linear=confusion_matrix(Y_test, Y_pred_SVM_linear)
TN=cf_svm_linear[0][0]
FP=cf_svm_linear[0][1]
FN=cf_svm_linear[1][0]
TP=cf_svm_linear[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_SVM_sigmoid),
   annot=True,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_svm_sigmoid.png",dpi=800)
plt.show()
cf_svm_sigmoid=confusion_matrix(Y_test, Y_pred_SVM_sigmoid)
```

```
TN=cf_svm_sigmoid[0][0]
FP=cf_svm_sigmoid[0][1]
FN=cf_svm_sigmoid[1][0]
TP=cf_svm_sigmoid[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_SVM_rbf), annot
    =True,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_svm_rbf.png",dpi=800)
plt.show()
cf_svm_rbf=confusion_matrix(Y_test, Y_pred_SVM_rbf)
TN=cf_svm_rbf[0][0]
FP=cf_svm_rbf[0][1]
FN=cf_svm_rbf[1][0]
TP=cf_svm_rbf[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))
```

```
plot_roc_curves(fpr_linear, tpr_linear,fpr_rbf, tpr_rbf,
    fpr_sigmoid, tpr_sigmoid)

"""## Naive Bayes

### Final
"""

model_NB = GaussianNB()
model_NB.fit(X_train, Y_train)
Y_pred_NB = model_NB.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_NB))
print('Accuracy:',metrics.accuracy_score(Y_test, Y_pred_NB))
print('Recall:',metrics.recall_score(Y_test, Y_pred_NB))
print('Precision:',metrics.precision_score(Y_test, Y_pred_NB)
    )

probs_nb = model_NB.predict_proba(X_test)
probs_nb = probs_nb[:,1]
auc_nb = roc_auc_score(Y_test, probs_nb)
fpr_nb, tpr_nb, thresholds_nb = roc_curve(Y_test, probs_nb)
plot_roc_curve(fpr_nb, tpr_nb,auc_nb)
oid = np.argmax(np.abs(tpr_nb - fpr_nb))
print('ROC:',auc_nb,'threshold:',thresholds_nb[oid],'TPR:',
    tpr_nb[oid],'FPR:',fpr_nb[oid])

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_NB), annot=True
    ,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_nb.png",dpi=800)
plt.show()
cf_nb=confusion_matrix(Y_test, Y_pred_NB)
TN=cf_nb[0][0]
FP=cf_nb[0][1]
```

```
FN=cf_nb[1][0]
TP=cf_nb[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

"""## Random Forest

### Par metros
"""

error_train_RM_gini = []
error_val_RM_gini = []
error_oob_RM_gini = []

N = range(1,702,25)
for i in N:
    model_RM_gini = RandomForestClassifier(n_estimators=i,
       random_state=0,oob_score=True,criterion='gini')
    model_RM_gini.fit(X_train, Y_train)
    Y_pred_train_RM_gini = model_RM_gini.predict(X_train)
    Y_pred_val_RM_gini = model_RM_gini.predict(X_val)
    Y_pred_test_RM_gini = model_RM_gini.predict(X_test)
    error_train_RM_gini.append(metrics.zero_one_loss(Y_train,
        Y_pred_train_RM_gini))
    error_val_RM_gini.append(metrics.zero_one_loss(Y_val,
       Y_pred_val_RM_gini))
    error_oob_RM_gini.append(1-model_RM_gini.oob_score_)
```

```
error_train_RM_ent = []
error_val_RM_ent = []
error_oob_RM_ent = []

N = range(1,702,25)
for i in N:
    model_RM_ent = RandomForestClassifier(n_estimators=i,
        random_state=0,oob_score=True,criterion='entropy')
    model_RM_ent.fit(X_train, Y_train)
    Y_pred_train_RM_ent = model_RM_ent.predict(X_train)
    Y_pred_val_RM_ent = model_RM_ent.predict(X_val)
    Y_pred_test_RM_ent = model_RM_ent.predict(X_test)
    error_train_RM_ent.append(metrics.zero_one_loss(Y_train,
        Y_pred_train_RM_ent))
    error_val_RM_ent.append(metrics.zero_one_loss(Y_val,
        Y_pred_val_RM_ent))
    error_oob_RM_ent.append(1-model_RM_ent.oob_score_)

plt.figure(figsize=(5,5))
plt.plot(N, error_train_RM_gini,label='Error entrenamiento')
plt.plot(N, error_val_RM_gini, label='Error validaci n')
plt.plot(N, error_oob_RM_gini,label='Error OOB')
plt.legend()
plt.axvline(x=475, color='grey', linestyle='--')
plt.savefig("ntrees_rm_gini.png",dpi=800)
plt.show()

plt.figure(figsize=(5,5))
plt.plot(N, error_train_RM_ent,label='Error entrenamiento')
plt.plot(N, error_val_RM_ent, label='Error validaci n')
plt.plot(N, error_oob_RM_ent,label='Error OOB')
plt.legend()
plt.axvline(x=400, color='grey', linestyle='--')
plt.savefig("ntrees_rm_ent.png",dpi=800)
plt.show()
```

```
"""### Final"""

model_RM_gini = RandomForestClassifier(n_estimators=475,
    random_state=0,criterion='gini')
model_RM_gini.fit(X_train, Y_train)
Y_pred_RM_gini = model_RM_gini.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_RM_gini))
print('Accuracy:',metrics.accuracy_score(Y_test,
    Y_pred_RM_gini))
print('Recall:',metrics.recall_score(Y_test, Y_pred_RM_gini))
print('Precision:',metrics.precision_score(Y_test,
    Y_pred_RM_gini))

model_RM_ent = RandomForestClassifier(n_estimators=400,
    random_state=0,criterion='entropy')
model_RM_ent.fit(X_train, Y_train)
Y_pred_RM_ent = model_RM_ent.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_RM_ent))
print('Accuracy:',metrics.accuracy_score(Y_test,
    Y_pred_RM_ent))
print('Recall:',metrics.recall_score(Y_test, Y_pred_RM_ent))
print('Precision:',metrics.precision_score(Y_test,
    Y_pred_RM_ent))

probs_gini = model_RM_gini.predict_proba(X_test)
probs_gini = probs_gini[:,1]
auc_gini = roc_auc_score(Y_test, probs_gini)
fpr_gini, tpr_gini, thresholds_gini = roc_curve(Y_test,
    probs_gini)
plot_roc_curve(fpr_gini, tpr_gini,auc_gini)
oid = np.argmax(np.abs(tpr_gini - fpr_gini))
print('ROC:',auc_gini,'threshold:',thresholds_gini[oid],'TPR
    :',tpr_gini[oid],'FPR:',fpr_gini[oid])

probs_ent = model_RM_ent.predict_proba(X_test)
probs_ent = probs_ent[:,1]
```

```
auc_ent = roc_auc_score(Y_test, probs_ent)
fpr_ent, tpr_ent, thresholds_ent = roc_curve(Y_test,
    probs_ent)
plot_roc_curve(fpr_ent, tpr_ent,auc_ent)
oid = np.argmax(np.abs(tpr_ent - fpr_ent))
print('ROC:',auc_ent,'threshold:',thresholds_ent[oid],'TPR:',
    tpr_ent[oid],'FPR:',fpr_ent[oid])

plot_roc_curves_rm(fpr_gini,tpr_gini,fpr_ent,tpr_ent)

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_RM_gini), annot
    =True,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_rm_gini.png",dpi=800)
plt.show()
cf_nb=confusion_matrix(Y_test, Y_pred_RM_gini)
TN=cf_nb[0][0]
FP=cf_nb[0][1]
FN=cf_nb[1][0]
TP=cf_nb[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_RM_ent), annot=
    True,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
```

```
p.figure.savefig("cm_rm_entropy.png",dpi=800)
plt.show()
cf_nb=confusion_matrix(Y_test, Y_pred_RM_ent)
TN=cf_nb[0][0]
FP=cf_nb[0][1]
FN=cf_nb[1][0]
TP=cf_nb[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

"""## Gradient Boost

### Par metros
"""

error_train_boost = []
error_val_boost = []

N = range(1,1002,25)
for i in N:
    model_GM = GradientBoostingClassifier(n_estimators=i,
       random_state=0,loss='exponential',criterion='mse')
    model_GM.fit(X_train, Y_train)
    Y_pred_train_boost = model_GM.predict(X_train)
    Y_pred_val_boost = model_GM.predict(X_val)
    Y_pred_test_boost = model_GM.predict(X_test)
    error_train_boost.append(1-metrics.accuracy_score(Y_train
       , Y_pred_train_boost))
```

```
    error_val_boost.append(1-metrics.accuracy_score(Y_val,
        Y_pred_val_boost))

plt.plot(N, error_train_boost,label='Error entrenamiento')
plt.plot(N, error_val_boost, label='Error validaci n')
plt.legend()
plt.axvline(x=630, color='grey', linestyle='--')
plt.savefig("ntrees_boost.png",dpi=800)
plt.show()

"""### Final"""

model_GB = GradientBoostingClassifier(random_state = 0,
                                      tol=0.00001,
                                      loss='exponential',
                                      criterion='mse',
                                      n_estimators=630)
model_GB.fit(X_train, Y_train)
Y_pred_GB = model_GB.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_GB))
print('Accuracy:',metrics.accuracy_score(Y_test, Y_pred_GB))
print('Recall:',metrics.recall_score(Y_test, Y_pred_GB))
print('Precision:',metrics.precision_score(Y_test, Y_pred_GB)
    )

probs_boost = model_GB.predict_proba(X_test)
probs_boost = probs_boost[:,1]
auc_boost = roc_auc_score(Y_test, probs_boost)
fpr_boost, tpr_boost, thresholds_boost = roc_curve(Y_test,
    probs_boost)
plot_roc_curve(fpr_boost, tpr_boost,auc_boost)
oid = np.argmax(np.abs(tpr_boost - fpr_boost))
print('ROC:',auc_boost,'threshold:',thresholds_boost[oid],'
    TPR:',tpr_boost[oid],'FPR:',fpr_boost[oid])

plt.figure(figsize=(3,3))
```

```
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_GB), annot=True
    ,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
#p.figure.savefig("cm_boost.png",dpi=800)
plt.show()
cf_nb=confusion_matrix(Y_test, Y_pred_GB)
TN=cf_nb[0][0]
FP=cf_nb[0][1]
FN=cf_nb[1][0]
TP=cf_nb[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))
```

B.0.2. Textos con stop words y textos no lematizados

```
import pandas as pd
import numpy as np
import re
import string
import functools
import matplotlib.pyplot as plt
import nltk
import textblob
import spacy
import seaborn as sns

from collections import Counter
from nltk.tokenize import word_tokenize
```

```
from sw import palabras_parada
from sklearn import metrics
from nltk.corpus import stopwords
from wordcloud import WordCloud
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report,
   confusion_matrix, plot_confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from imblearn.over_sampling import SMOTE
from sklearn.feature_extraction.text import CountVectorizer,
   TfidfVectorizer
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

nltk.download("stopwords")
nlp_md = spacy.load("es_core_news_md")
sw=stopwords.words('spanish')+palabras_parada
sns.set_style("white")
sns.set_context("paper")

"""# Funciones"""

def remover_links(x):
    l=re.compile('https:\S*')
    return l.sub('',x.lower())
def remover_puntc(x):
    #p=re.compile('[.;,  ?! _ |"()-:]')
    p=re.compile('[^\w\s]')
    return p.sub(' ',x.lower()) # remplaza por espacio
def remover_caract_ext(x):
```

```
    c=re.compile('<\S*>')
    return c.sub('',x.lower())
def remover_carac_espacio(x):
    e=re.compile('(\r)|(\n)')
    return e.sub('',x.lower())
def remover_num(x):
    n=re.compile('[0-9]*')
    return n.sub('',x.lower())
def remover_etiq(x):
    e=re.compile('@\S*')
    return e.sub('',x.lower())
def remover_hash(x):
    h=re.compile('#\S*')
    return h.sub('',x.lower())
def remover_single(x):
    h=re.compile('\s\w\s')
    return h.sub(' ',x.lower())
def remover_spaces(x):
    return " ".join(x.split())

def remover(x):
    aux=x
    aux = remover_etiq(aux)
    aux = remover_links(aux)
    aux = remover_hash(aux)
    aux = remover_caract_ext(aux)
    aux = remover_carac_espacio(aux)
    aux = remover_puntc(aux)
    aux = remover_num(aux)
    aux = remover_single(aux)
    aux = remover_spaces(aux)
    return aux.strip()

def lematizar_md(x):
    return ' '.join([tok.lemma_.lower() for tok in nlp_md(x)
       ])
```

```
def no_sw(x):
    return ' '.join([t for t in x.split() if t not in sw])

def plot_roc_curve(fpr, tpr,auc):
    plt.plot(fpr, tpr, color='orange')
    plt.plot([0, 1], [0, 1], color=(0.12156862745098039,
       0.4666666666666667, 0.7058823529411765), linestyle
       ='--')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.show()

def plot_roc_curves(fpr1,tpr1,fpr2,tpr2,fpr3,tpr3):
    fig = plt.figure()
    palette = plt.get_cmap('tab10')
    plt.plot(fpr1, tpr1,color=palette(2),label='svm lineal')
    plt.plot(fpr2, tpr2,color=palette(0),label='svm rbf')
    plt.plot(fpr3, tpr3,color=palette(1), label='svm sigmoide
       ')
    plt.plot([0, 1], [0, 1], color='silver', linestyle='--')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.legend()
    fig.savefig('svm_roc_no', dpi=800)
    plt.show()

def plot_roc_curves_rm(fpr1,tpr1,fpr2,tpr2):
    fig = plt.figure()
    palette = plt.get_cmap('tab10')
    plt.plot(fpr1, tpr1,color=palette(0),label='rm gini')
    plt.plot(fpr2, tpr2,color=palette(1),label='rm entropy')
    plt.plot([0, 1], [0, 1], color='silver', linestyle='--')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.legend()
```

```
    fig.savefig('rm_roc_no', dpi=800)
    plt.show()

def count_words(texto):
    return len(texto.split())

"""#Carga de Datos"""

data = pd.read_csv('base_noticias.csv')
data

"""# EDA textos"""

plt.figure(figsize=(4,5))
v=len(data['Tipo'][data['Tipo']==0])
f=len(data['Tipo'][data['Tipo']==1])
p=sns.countplot(x = 'Tipo', data = data)
p.set(xlabel='Tipo de noticia', ylabel='Frecuencia')
p.figure.savefig("data_dist.png",dpi=800)
'Verdaderas: {:.2f}% Falsas: {:.2f}%'.format(v/(v+f), f/(v+f)
    )

p=sns.countplot(y = 'Pagina', data = data,order =data['Pagina
    '].value_counts().index)
p.set(xlabel='Total de noticias', ylabel='')
p.figure.savefig("fuentes.png")
plt.show()

texto = data['Noticia']
texto

textos_len = list(map(count_words,texto))
base_len = pd.DataFrame({'Longitud':textos_len,'Tipo':data['
    Tipo']})

plt.figure(figsize=(4,6))
```

```
p=sns.boxplot(x='Tipo', y='Longitud', data=base_len,linewidth
    =0.75)
p.set(ylabel='N mero de palabras')
p.figure.savefig("text_boxplot.png",dpi=800)
plt.show()

plt.figure(figsize=(10,5))
p=sns.displot(x="Longitud", hue="Tipo",data=base_len, legend=
    False)
p.set(xlabel='N mero de palabras',ylabel='')
p.despine(left=True)
p.savefig("dist_text.png",dpi=800)
plt.show()

base_len.groupby('Tipo').agg({'Longitud': ['mean','std']})

"""# Procesamiento de texto

## Limpiar puntuaci n, hashtags, etiquetas, links.
"""

texto = list(map(remover,texto))

"""## Lematizar"""

texto = list(map(lematizar_md,texto))

texto = list(map(remover,texto))

"""## Stopwords"""

texto_sin_sw = texto

"""# Embedding TD-IDF"""

vector = TfidfVectorizer()
```

```
vector.fit(texto_sin_sw)
data_pv_dm=vector.transform(texto_sin_sw).toarray()
pd.DataFrame(data_pv_dm)

"""# Entrenamiento

## Conjuntos de datos
"""

X=data_pv_dm
Y=np.array(data['Tipo'])
X_train_sm, X_test, Y_train_sm, Y_test = train_test_split(X,
    Y, test_size = 0.2,random_state=123)
X_val, X_test, Y_val, Y_test= train_test_split(X_test, Y_test
    , test_size=0.5, random_state=111)
smt = SMOTE(random_state=0,k_neighbors=5)
X_train, Y_train = smt.fit_sample(X_train_sm, Y_train_sm)

X_aum = pd.concat([pd.DataFrame(X_train),pd.DataFrame(X_val),
    pd.DataFrame(X_test)], axis=0, ignore_index=True)
Y_aum = pd.concat([pd.DataFrame(Y_train),pd.DataFrame(Y_val),
    pd.DataFrame(Y_test)], axis=0, ignore_index=True)
data_aum = pd.concat([X_aum,Y_aum], axis=1, ignore_index=True
    )
data_aum = data_aum.rename(columns={9953:'Tipo'})

plt.figure(figsize=(4,5))
v=len(data_aum['Tipo'][data_aum['Tipo']==0])
f=len(data_aum['Tipo'][data_aum['Tipo']==1])
p=sns.countplot(x = 'Tipo', data = data_aum)
p.set(xlabel='Tipo de noticia', ylabel='Frecuencia')
p.figure.savefig("data_dist_aum.png",dpi=800)
'Verdaderas: {:.2f}% Falsas: {:.2f}%'.format(v/(v+f), f/(v+f)
    )

"""## SVM
```

```
### Flexibilidad
"""

error = []
C = [0.01,0.5,1,5,25,50,100]
for i in C:
    model_SVM = SVC(kernel='sigmoid',tol=0.0001,C=i,
        random_state=0)
    model_SVM.fit(X_train, Y_train)
    Y_pred = model_SVM.predict(X_val)
    error.append(metrics.zero_one_loss(Y_val, Y_pred))

plt.figure(figsize=(3,5))
results_sigmoid = pd.DataFrame({'C
   ':['0.01','0.5','1','5','25','50','100'],'Error':error})
p=sns.barplot(x='C',y = 'Error', data = results_sigmoid,order
    =results_sigmoid.sort_values('Error',ascending=False).C,
   color=(0.12156862745098039, 0.4666666666666667,
   0.7058823529411765))
p.set(ylabel='Loss')
plt.savefig("svm_sigmoid_C_no.png",dpi=800)
plt.show()

error = []
C = [0.01,0.5,1,5,25,50,100]
for i in C:
    model_SVM = SVC(kernel='rbf',tol=0.0001,C=i,random_state
        =0)
    model_SVM.fit(X_train, Y_train)
    Y_pred = model_SVM.predict(X_val)
    error.append(metrics.zero_one_loss(Y_val, Y_pred))

plt.figure(figsize=(3,5))
results_rbf = pd.DataFrame({'C
   ':['0.01','0.5','1','5','25','50','100'],'Error':error})
```

```
p=sns.barplot(x='C',y = 'Error', data = results_rbf,order =
    results_rbf.sort_values('Error',ascending=False).C,color
    =(0.12156862745098039, 0.4666666666666667,
    0.7058823529411765))
p.set(ylabel='Loss')
plt.savefig("svm_rbf_C_no.png",dpi=800)
plt.show()

error = []
C = [0.01,0.5,1,5,25,50,100]
for i in C:
    model_SVM = SVC(kernel='linear',tol=0.0001,C=i,
        random_state=0)
    model_SVM.fit(X_train, Y_train)
    Y_pred = model_SVM.predict(X_val)
    error.append(metrics.zero_one_loss(Y_val, Y_pred))

plt.figure(figsize=(3,5))
results_linear = pd.DataFrame({'C
    ':['0.01','0.5','1','5','25','50','100'],'Error':error})
p=sns.barplot(x='C',y = 'Error', data = results_linear,order
    =results_linear.sort_values('Error',ascending=False).C,
    color=(0.12156862745098039, 0.4666666666666667,
    0.7058823529411765))
p.set(ylabel='Loss')
plt.savefig("svm_linear_C_no.png",dpi=800)
plt.show()

"""### Final"""

model_SVM_sigmoid = SVC(kernel='sigmoid',tol=0.00001,
    random_state=0,probability=True,C=1)
model_SVM_sigmoid.fit(X_train, Y_train)
Y_pred_SVM_sigmoid = model_SVM_sigmoid.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_SVM_sigmoid))
```

```
print('Accuracy:',metrics.accuracy_score(Y_test,
    Y_pred_SVM_sigmoid))
print('Recall:',metrics.recall_score(Y_test,
    Y_pred_SVM_sigmoid))
print('Precision:',metrics.precision_score(Y_test,
    Y_pred_SVM_sigmoid))

model_SVM_rbf = SVC(kernel='rbf',tol=0.00001,random_state=0,
    probability=True,C=5)
model_SVM_rbf.fit(X_train, Y_train)
Y_pred_SVM_rbf = model_SVM_rbf.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_SVM_rbf))
print('Accuracy:',metrics.accuracy_score(Y_test,
    Y_pred_SVM_rbf))
print('Recall:',metrics.recall_score(Y_test, Y_pred_SVM_rbf))
print('Precision:',metrics.precision_score(Y_test,
    Y_pred_SVM_rbf))

model_SVM_linear = SVC(kernel='linear',tol=0.00001,
    random_state=0,probability=True,C=5)
model_SVM_linear.fit(X_train, Y_train)
Y_pred_SVM_linear = model_SVM_linear.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_SVM_linear))
print('Accuracy:',metrics.accuracy_score(Y_test,
    Y_pred_SVM_linear))
print('Recall:',metrics.recall_score(Y_test,
    Y_pred_SVM_linear))
print('Precision:',metrics.precision_score(Y_test,
    Y_pred_SVM_linear))

probs_sigmoid = model_SVM_sigmoid.predict_proba(X_test)
probs_sigmoid = probs_sigmoid[:,1]
auc_sigmoid = roc_auc_score(Y_test, probs_sigmoid)
fpr_sigmoid, tpr_sigmoid, thresholds_sigmoid = roc_curve(
    Y_test, probs_sigmoid)
plot_roc_curve(fpr_sigmoid, tpr_sigmoid,auc_sigmoid)
```

```
oid = np.argmax(np.abs(tpr_sigmoid - fpr_sigmoid))
print('ROC:',auc_sigmoid,'threshold:',thresholds_sigmoid[oid
    ],'TPR:',tpr_sigmoid[oid],'FPR:',fpr_sigmoid[oid])

probs_rbf = model_SVM_rbf.predict_proba(X_test)
probs_rbf = probs_rbf[:,1]
auc_rbf = roc_auc_score(Y_test, probs_rbf)
fpr_rbf, tpr_rbf, thresholds_rbf = roc_curve(Y_test,
    probs_rbf)
plot_roc_curve(fpr_rbf, tpr_rbf,auc_rbf)
oid = np.argmax(np.abs(tpr_rbf - fpr_rbf))
print('ROC:',auc_rbf,'threshold:',thresholds_rbf[oid],'TPR:',
    tpr_rbf[oid],'FPR:',fpr_rbf[oid])

probs_linear = model_SVM_linear.predict_proba(X_test)
probs_linear = probs_linear[:,1]
auc_linear = roc_auc_score(Y_test, probs_linear)
fpr_linear, tpr_linear, thresholds_linear = roc_curve(Y_test,
     probs_linear)
plot_roc_curve(fpr_linear, tpr_linear,auc_linear)
oid = np.argmax(np.abs(tpr_linear - fpr_linear))
print('ROC:',auc_linear,'threshold:',thresholds_linear[oid],'
    TPR:',tpr_linear[oid],'FPR:',fpr_linear[oid])

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_SVM_linear),
    annot=True,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_svm_linear_no.png",dpi=800)
plt.show()
cf_svm_linear=confusion_matrix(Y_test, Y_pred_SVM_linear)
TN=cf_svm_linear[0][0]
FP=cf_svm_linear[0][1]
FN=cf_svm_linear[1][0]
TP=cf_svm_linear[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
```

```
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_SVM_sigmoid),
    annot=True,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_svm_sigmoid_no.png",dpi=800)
plt.show()
cf_svm_sigmoid=confusion_matrix(Y_test, Y_pred_SVM_sigmoid)
TN=cf_svm_sigmoid[0][0]
FP=cf_svm_sigmoid[0][1]
FN=cf_svm_sigmoid[1][0]
TP=cf_svm_sigmoid[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_SVM_rbf), annot
    =True,fmt='g',cmap='Blues',cbar = False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
```

```
p.figure.savefig("cm_svm_rbf_no.png",dpi=800)
plt.show()
cf_svm_rbf=confusion_matrix(Y_test, Y_pred_SVM_rbf)
TN=cf_svm_rbf[0][0]
FP=cf_svm_rbf[0][1]
FN=cf_svm_rbf[1][0]
TP=cf_svm_rbf[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

plot_roc_curves(fpr_linear, tpr_linear,fpr_rbf, tpr_rbf,
    fpr_sigmoid, tpr_sigmoid)

"""## Naive Bayes

### Final
"""

model_NB = GaussianNB()
model_NB.fit(X_train, Y_train)
Y_pred_NB = model_NB.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_NB))
print('Accuracy:',metrics.accuracy_score(Y_test, Y_pred_NB))
print('Recall:',metrics.recall_score(Y_test, Y_pred_NB))
print('Precision:',metrics.precision_score(Y_test, Y_pred_NB)
    )

probs_nb_no = model_NB.predict_proba(X_test)
```

```
probs_nb_no = probs_nb_no[:,1]
auc_nb_no = roc_auc_score(Y_test, probs_nb_no)
fpr_nb_no, tpr_nb_no, thresholds_nb_no = roc_curve(Y_test,
    probs_nb_no)
plot_roc_curve(fpr_nb_no, tpr_nb_no,auc_nb_no)
oid = np.argmax(np.abs(tpr_nb_no - fpr_nb_no))
print('ROC:',auc_nb_no,'threshold:',thresholds_nb_no[oid],'
    TPR:',tpr_nb_no[oid],'FPR:',fpr_nb_no[oid])

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_NB), annot=True
    ,fmt='g',cmap='Blues',cbar = False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_nb_no.png",dpi=800)
plt.show()
cf_nb=confusion_matrix(Y_test, Y_pred_NB)
TN=cf_nb[0][0]
FP=cf_nb[0][1]
FN=cf_nb[1][0]
TP=cf_nb[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

"""## Random Forest

### Par metros
"""
```

```
error_train_RM_gini = []
error_val_RM_gini = []
error_oob_RM_gini = []

N = range(1,702,25)
for i in N:
    model_RM_gini = RandomForestClassifier(n_estimators=i,
        random_state=0,oob_score=True,criterion='gini')
    model_RM_gini.fit(X_train, Y_train)
    Y_pred_train_RM_gini = model_RM_gini.predict(X_train)
    Y_pred_val_RM_gini = model_RM_gini.predict(X_val)
    Y_pred_test_RM_gini = model_RM_gini.predict(X_test)
    error_train_RM_gini.append(metrics.zero_one_loss(Y_train,
        Y_pred_train_RM_gini))
    error_val_RM_gini.append(metrics.zero_one_loss(Y_val,
        Y_pred_val_RM_gini))
    error_oob_RM_gini.append(1-model_RM_gini.oob_score_)

error_train_RM_ent = []
error_val_RM_ent = []
error_oob_RM_ent = []

N = range(1,702,25)
for i in N:
    model_RM_ent = RandomForestClassifier(n_estimators=i,
        random_state=0,oob_score=True,criterion='entropy')
    model_RM_ent.fit(X_train, Y_train)
    Y_pred_train_RM_ent = model_RM_ent.predict(X_train)
    Y_pred_val_RM_ent = model_RM_ent.predict(X_val)
    Y_pred_test_RM_ent = model_RM_ent.predict(X_test)
    error_train_RM_ent.append(metrics.zero_one_loss(Y_train,
        Y_pred_train_RM_ent))
    error_val_RM_ent.append(metrics.zero_one_loss(Y_val,
        Y_pred_val_RM_ent))
    error_oob_RM_ent.append(1-model_RM_ent.oob_score_)
```

```
plt.figure(figsize=(5,5))
plt.plot(N, error_train_RM_gini,label='Error entrenamiento')
plt.plot(N, error_val_RM_gini, label='Error validaci n')
plt.plot(N, error_oob_RM_gini,label='Error OOB')
plt.legend()
plt.axvline(x=450, color='grey', linestyle='--')
plt.savefig("ntrees_rm_gini_no.png",dpi=800)
plt.show()

plt.figure(figsize=(5,5))
plt.plot(N, error_train_RM_ent,label='Error entrenamiento')
plt.plot(N, error_val_RM_ent, label='Error validaci n')
plt.plot(N, error_oob_RM_ent,label='Error OOB')
plt.legend()
plt.axvline(x=350, color='grey', linestyle='--')
plt.savefig("ntrees_rm_ent_no.png",dpi=800)
plt.show()

"""### Final"""

model_RM_gini = RandomForestClassifier(n_estimators=450,
    random_state=0,criterion='gini')
model_RM_gini.fit(X_train, Y_train)
Y_pred_RM_gini = model_RM_gini.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_RM_gini))
print('Accuracy:',metrics.accuracy_score(Y_test,
    Y_pred_RM_gini))
print('Recall:',metrics.recall_score(Y_test, Y_pred_RM_gini))
print('Precision:',metrics.precision_score(Y_test,
    Y_pred_RM_gini))

model_RM_ent = RandomForestClassifier(n_estimators=350,
    random_state=0,criterion='entropy')
model_RM_ent.fit(X_train, Y_train)
Y_pred_RM_ent = model_RM_ent.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_RM_ent))
```

```
print('Accuracy:',metrics.accuracy_score(Y_test,
    Y_pred_RM_ent))
print('Recall:',metrics.recall_score(Y_test, Y_pred_RM_ent))
print('Precision:',metrics.precision_score(Y_test,
    Y_pred_RM_ent))

probs_gini = model_RM_gini.predict_proba(X_test)
probs_gini = probs_gini[:,1]
auc_gini = roc_auc_score(Y_test, probs_gini)
fpr_gini, tpr_gini, thresholds_gini = roc_curve(Y_test,
    probs_gini)
plot_roc_curve(fpr_gini, tpr_gini,auc_gini)
oid = np.argmax(np.abs(tpr_gini - fpr_gini))
print('ROC:',auc_gini,'threshold:',thresholds_gini[oid],'TPR
    :',tpr_gini[oid],'FPR:',fpr_gini[oid])

probs_ent = model_RM_ent.predict_proba(X_test)
probs_ent = probs_ent[:,1]
auc_ent = roc_auc_score(Y_test, probs_ent)
fpr_ent, tpr_ent, thresholds_ent = roc_curve(Y_test,
    probs_ent)
plot_roc_curve(fpr_ent, tpr_ent,auc_ent)
oid = np.argmax(np.abs(tpr_ent - fpr_ent))
print('ROC:',auc_ent,'threshold:',thresholds_ent[oid],'TPR:',
    tpr_ent[oid],'FPR:',fpr_ent[oid])

plot_roc_curves_rm(fpr_gini,tpr_gini,fpr_ent,tpr_ent)

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_RM_gini), annot
    =True,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_rm_gini_no.png",dpi=800)
plt.show()
cf_nb=confusion_matrix(Y_test, Y_pred_RM_gini)
TN=cf_nb[0][0]
```

```
FP=cf_nb[0][1]
FN=cf_nb[1][0]
TP=cf_nb[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_RM_ent), annot=
    True,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
p.figure.savefig("cm_rm_entropy_no.png",dpi=800)
plt.show()
cf_nb=confusion_matrix(Y_test, Y_pred_RM_ent)
TN=cf_nb[0][0]
FP=cf_nb[0][1]
FN=cf_nb[1][0]
TP=cf_nb[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))
```

```
"""## Gradient Boost

### Par metros
"""

error_train_boost = []
error_val_boost = []

N = range(1,1002,25)
for i in N:
    model_GM = GradientBoostingClassifier(n_estimators=i,
       random_state=0,loss='exponential',criterion='mse')
    model_GM.fit(X_train, Y_train)
    Y_pred_train_boost = model_GM.predict(X_train)
    Y_pred_val_boost = model_GM.predict(X_val)
    Y_pred_test_boost = model_GM.predict(X_test)
    error_train_boost.append(1-metrics.accuracy_score(Y_train
       , Y_pred_train_boost))
    error_val_boost.append(1-metrics.accuracy_score(Y_val,
       Y_pred_val_boost))

plt.plot(N, error_train_boost,label='Error entrenamiento')
plt.plot(N, error_val_boost, label='Error validaci n')
plt.legend()
plt.axvline(x=900, color='grey', linestyle='--')
plt.savefig("ntrees_boost_no.png",dpi=800)
plt.show()

"""### Final"""

model_GB = GradientBoostingClassifier(random_state = 0,
                                      tol=0.00001,
                                      loss='exponential',
                                      criterion='mse',
                                      n_estimators=900)
model_GB.fit(X_train, Y_train)
```

```
Y_pred_GB = model_GB.predict(X_test)
print(confusion_matrix(Y_test, Y_pred_GB))
print('Accuracy:',metrics.accuracy_score(Y_test, Y_pred_GB))
print('Recall:',metrics.recall_score(Y_test, Y_pred_GB))
print('Precision:',metrics.precision_score(Y_test, Y_pred_GB)
    )

probs_boost = model_GB.predict_proba(X_test)
probs_boost = probs_boost[:,1]
auc_boost = roc_auc_score(Y_test, probs_boost)
fpr_boost, tpr_boost, thresholds_boost = roc_curve(Y_test,
    probs_boost)
plot_roc_curve(fpr_boost, tpr_boost,auc_boost)
oid = np.argmax(np.abs(tpr_boost - fpr_boost))
print('ROC:',auc_boost,'threshold:',thresholds_boost[oid],'
    TPR:',tpr_boost[oid],'FPR:',fpr_boost[oid])

plt.figure(figsize=(3,3))
p=sns.heatmap(confusion_matrix(Y_test, Y_pred_GB), annot=True
    ,fmt='g',cmap='Blues',cbar=False)
p.set(xlabel='clase predicha', ylabel='clase verdadera')
#p.figure.savefig("cm_boost.png",dpi=800)
plt.show()
cf_nb=confusion_matrix(Y_test, Y_pred_GB)
TN=cf_nb[0][0]
FP=cf_nb[0][1]
FN=cf_nb[1][0]
TP=cf_nb[1][1]
print('Err:',100*(FP+FN)/(FP+FN+TN+TP))
print('Acc:',100*(TP+TN)/(FP+FN+TN+TP))
print('precF:',100*(TP)/(TP+FP))
print('precR:',100*(TN)/(TN+FN))
print('TPR:',100*(TP)/(TP+FN))
print('TFR:',100*(TN)/(TN+FP))
print('FNR:',100*(FN)/(TP+FN))
print('FPR:',100*(FP)/(TN+FP))
```

```
print('F-F:',(2*TP)/((2*TP)+FP+FN))
print('F-R:',(2*TN)/((2*TN)+FN+FP))
```